


2011

## Multiagent Learning Through Indirect Encoding

David B. D'Ambrosio  
*University of Central Florida*

 Part of the [Electrical and Electronics Commons](#)  
Find similar works at: <https://stars.library.ucf.edu/etd>  
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact [STARS@ucf.edu](mailto:STARS@ucf.edu).

---

### STARS Citation

D'Ambrosio, David B., "Multiagent Learning Through Indirect Encoding" (2011). *Electronic Theses and Dissertations, 2004-2019*. 2027.  
<https://stars.library.ucf.edu/etd/2027>

# MULTIAGENT LEARNING THROUGH INDIRECT ENCODING

by

DAVID B. D'AMBROSIO  
B.S. Florida Atlantic University, 2004  
M.S. University of Central Florida, 2008

A dissertation submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy  
in the Department of Electrical Engineering and Computer Science  
in the College of Engineering and Computer Science  
at the University of Central Florida  
Orlando, Florida

Spring Term  
2011

Major Professor:  
Kenneth O. Stanley

© 2011 by David B. D'Ambrosio

## ABSTRACT

Designing a system of multiple, heterogeneous agents that cooperate to achieve a common goal is a difficult task, but it is also a common real-world problem. Multiagent learning addresses this problem by training the team to cooperate through a learning algorithm. However, most traditional approaches treat multiagent learning as a combination of multiple single-agent learning problems. This perspective leads to many inefficiencies in learning such as the *problem of reinvention*, whereby fundamental skills and policies that all agents should possess must be rediscovered independently for each team member. For example, in soccer, all the players know how to pass and kick the ball, but a traditional algorithm has no way to share such vital information because it has no way to relate the policies of agents to each other.

In this dissertation a new approach to multiagent learning that seeks to address these issues is presented. This approach, called *multiagent HyperNEAT*, represents teams as a *pattern of policies* rather than individual agents. The main idea is that an agent's location within a canonical team layout (such as a soccer team at the start of a game) tends to dictate its role within that team, called the *policy geometry*. For example, as soccer positions move from goal to center they become more offensive and less defensive, a concept that is compactly represented as a pattern.

The first major contribution of this dissertation is a new method for evolving neural network controllers called HyperNEAT, which forms the foundation of the second contribution and primary focus of this work, multiagent HyperNEAT. Multiagent learning in this dissertation is investigated in predator-prey, room-clearing, and patrol domains, providing a real-world context for the approach. Interestingly, because the teams in multiagent HyperNEAT are represented as patterns they can scale up to an infinite number of multiagent policies that can be sampled from the policy geometry as needed. Thus the third contribution is a method for teams trained with multiagent HyperNEAT to dynamically scale their size *without further learning*. Fourth, the capabilities to both learn and scale in multiagent HyperNEAT are compared to the traditional multiagent SARSA( $\lambda$ ) approach in a comprehensive study. The fifth contribution is a method for efficiently learning and encoding multiple policies for each agent on a team to facilitate learning in multi-task domains. Finally, because there is significant interest in practical applications of multiagent learning, multiagent HyperNEAT is tested in a real-world military patrolling application with actual Khepera III robots. The ultimate goal is to provide a new perspective on multiagent learning and to demonstrate the practical benefits of training heterogeneous, scalable multiagent teams through generative encoding.

*To my parents, Kathy and David, and to Monique Bouvier.*

## ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Kenneth O. Stanley who taught me how to write, introduced me to neuroevolution, and was a general source of inspiration throughout my Ph. D. career. Without him this dissertation would not exist.

Thanks to my other committee members Dr. Gita R. Sukthankar, and Dr. R. Paul Wiegand, and Dr. Annie S. Wu whose advice and comments further elevated the quality of this dissertation.

I would also like to thank my parents, who sacrificed so much so that I can be where I am now: my mother, for dedicating so much time to my education when I was young and my father, for working so hard for so long. I hope that this dissertation and my future endeavors can match what they have done for me. Also thanks to Monique Bouvier who was a constant source of inspiration and compassion.

Thanks to past and present members of the Evolutionary Complexity Research Group (Eplex) at University of Central Florida (<http://eplex.cs.ucf.edu>) for the inspirational discussions and valuable feedback. In particular, thanks to Joel Lehman and Sebastian Risi who worked with me in the later stages of my research. Without them this document would not be what it is. We spent many long, frustrating hours in the robot lab together; hopefully this dissertation is proof that it was not in vain.

Thanks also to DARPA for funding the majority of this research through grants HR0011-08-1-0020 and HR0011-09-1-0045 (Computer Science Study Group Phases I and II).

Finally, special thanks to Colin Green for his software package SharpNEAT, which served as the basis for the software that ran all the experiments in this dissertation: HyperSharpNEAT. Without his expertise and optimizations, my experiments would have taken much longer to run. Also thanks to those who downloaded HyperSharpNEAT and provided bug reports and feedback.

David B. D'Ambrosio

University of Central Florida

April 2011



## TABLE OF CONTENTS

<b>LIST OF FIGURES</b>	<b>xiii</b>
<b>LIST OF TABLES</b>	<b>xxxi</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Contributions	5
1.2 Outline	6
<b>CHAPTER 2 BACKGROUND</b>	<b>7</b>
2.1 Cooperative Multiagent Learning	7
2.1.1 Traditional Approaches	8
2.1.2 Alternative Techniques	11
2.2 NeuroEvolution of Augmenting Topologies (NEAT)	13
2.3 Generative and Developmental Systems	15
2.3.1 Compositional Pattern Producing Networks (CPPNs)	16
<b>CHAPTER 3 HYPERNEAT</b>	<b>21</b>

3.1	Mapping Spatial Patterns to Connectivity Patterns . . . . .	22
3.2	Producing Regular Connectivity Patterns . . . . .	24
3.3	Substrate Configuration . . . . .	26
3.4	Input and Output Placement . . . . .	28
3.5	Substrate Resolution . . . . .	30
3.6	Evolving Connective CPPNs . . . . .	32
<b>CHAPTER 4</b>	<b>SINGLE-AGENT HYPERNEAT . . . . .</b>	<b>34</b>
4.1	Food Gathering Experiment . . . . .	35
4.1.1	Sensor Placement . . . . .	37
4.1.2	Additional Geometric Bias . . . . .	37
4.1.3	Scaling . . . . .	38
4.2	Food Gathering Results . . . . .	39
4.2.1	Scaling Performance . . . . .	41
4.2.2	Repeating Motifs . . . . .	43
4.2.3	Scaling Analysis . . . . .	43
4.2.4	Evolving Motifs . . . . .	46
4.3	Single-Agent Experiment Discussion . . . . .	47
4.3.1	Substrate Scaling . . . . .	48

4.3.2	Substrate Configuration and Geometry . . . . .	48
<b>CHAPTER 5</b>	<b>APPROACH: MULTIAGENT HYPERNEAT . . . . .</b>	<b>50</b>
5.1	Pure Homogeneous Teams . . . . .	51
5.2	Teams on the Continuum of Heterogeneity . . . . .	52
5.3	Seeding . . . . .	55
<b>CHAPTER 6</b>	<b>MULTIAGENT PREDATOR-PREY EXPERIMENT . . . . .</b>	<b>58</b>
6.1	Predator-Prey Experiment . . . . .	58
6.1.1	Predators and Prey . . . . .	59
6.1.2	Homogeneous vs. Heterogeneous Policies . . . . .	61
6.1.3	Seeding . . . . .	62
6.1.4	Prey Formations . . . . .	63
6.2	Predator-Prey Results . . . . .	64
6.2.1	Training Performance . . . . .	64
6.2.2	Generalization . . . . .	67
6.2.3	Typical Behaviors . . . . .	68
6.3	Predator-Prey Discussion . . . . .	70
<b>CHAPTER 7</b>	<b>SCALABLE MULTIAGENT HYPERNEAT . . . . .</b>	<b>72</b>
7.1	Substrate Scaling . . . . .	74

7.1.1	Alternative Substrate . . . . .	74
7.1.2	Substrate Validation Experiment . . . . .	78
7.2	Scaling Experiments . . . . .	79
7.2.1	Predator-Prey Scaling Performance . . . . .	81
7.2.2	Room Clearing Experiment . . . . .	86
7.2.2.1	Scaling in Room Clearing . . . . .	88
7.2.2.2	Room Clearing Results . . . . .	89
7.3	Scaling Discussion . . . . .	91
<b>CHAPTER 8 COMPARISON WITH REINFORCEMENT LEARNING .</b>		<b>93</b>
8.1	Predator-Prey Experiment . . . . .	93
8.1.1	Predators and Prey . . . . .	94
8.1.2	Prey Formations . . . . .	95
8.1.3	Scaling . . . . .	98
8.1.4	Seeding . . . . .	99
8.1.5	Reinforcement Learning Parameters . . . . .	100
8.2	Comparison Results . . . . .	100
8.2.1	Post-training Scaling with Further Learning . . . . .	106
8.2.2	Typical Behaviors . . . . .	111

8.3	Comparison Discussion . . . . .	116
<b>CHAPTER 9 REAL-WORLD MULTI-TASK LEARNING . . . . .</b>		<b>121</b>
9.1	Situational Policy Geometry . . . . .	122
9.2	Patrol and Return Experiment . . . . .	123
9.2.1	Robots . . . . .	126
9.2.2	Environments . . . . .	128
9.3	Patrol Results . . . . .	131
9.4	Patrol Discussion . . . . .	133
<b>CHAPTER 10 DISCUSSION AND FUTURE WORK . . . . .</b>		<b>136</b>
10.1	Policy Geometry and the Continuum of Heterogeneity . . . . .	136
10.2	Scaling . . . . .	138
10.3	Future Work . . . . .	139
<b>CHAPTER 11 CONCLUSION . . . . .</b>		<b>141</b>
11.1	Contributions . . . . .	141
11.2	Conclusion . . . . .	144
<b>APPENDIX: PARAMETERS . . . . .</b>		<b>146</b>
<b>LIST OF REFERENC ES . . . . .</b>		<b>150</b>

## LIST OF FIGURES

1.1	<p>Role as a Function of Geometry. In the depicted soccer team, the defensive to offensive dimension of variation spans from left to right. The question marks represent a hypothetical new line of players, whose roles can be inferred from their positions. The main conclusion is that policies are distributed in a <i>pattern</i> in space. . . . .</p>	2
2.1	<p>CPPN Encoding. (a) The CPPN takes arguments <math>x</math> and <math>y</math>, which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of the CPPN, the result is a spatial pattern, which can be viewed as a phenotype whose genotype is the CPPN. (b) Internally, the CPPN is a graph that determines which functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. The CPPN in (b) actually produces the pattern in (a) . . . . .</p>	18

- 2.2 CPPN-generated Regularities. Spatial patterns exhibiting (a) bilateral symmetry, (b) imperfect symmetry, and (c) repetition with variation (notice the nexus of each repeated motif) are depicted. These patterns demonstrate that CPPNs effectively encode fundamental regularities of several different types. 20
- 3.1 Hypercube-based Geometric Connectivity Pattern Interpretation. A collection of nodes, called the *substrate*, is assigned coordinates that range from  $-1$  to  $1$  in all dimensions. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines in the substrate depicted in the figure represent a sample of connections that are queried. (2) For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. Thus, CPPNs can produce regular patterns of connections in space. . . . . 23
- 3.2 Connectivity Patterns Produced by Connective CPPNs. These patterns, produced through interactive evolution, exhibit several important connectivity motifs: (a) bilateral symmetry, (b) imperfect symmetry, (c) repetition, and (d) repetition with variation. That these fundamental motifs are compactly represented and easily produced suggests the power of this encoding. . . . . 25

3.3	Alternative Substrate Configurations. The two-dimensional grid configuration (a), similar to the one in figure 3.1, is only one of many potential substrate configurations. This figure also shows (b) a three-dimensional configuration of nodes centered at $(0,0,0)$ , (c) a “state-space sandwich” configuration in which a source sheet of neurons connects directly to a target sheet, and (d) a circular configuration. Different configurations are suited to problems with different geometric properties. . . . .	26
3.4	Regular ANN Input Order is Irrelevant. Even though the order of inputs to the ANN in (a) is correlated directly to the geometry of the front-facing robot sensors from which they are activated, traditional ANN learning algorithms such as backpropagation [RHW86] or traditional neuroevolution methods [SM02b, Yao99] are blind to such ordering. In fact, the arbitrary input order in (b) is identical from the perspective of the learning algorithm. Thus, counterintuitively, organizing inputs (or outputs) to respect task geometry provides no advantage with traditional ANN learning algorithms. . . . .	28



3.5	Placing Inputs and Outputs. A robot (a) is depicted with eight radar sensors along its circumference and eight motion effectors set at the same angle. In (b), the inputs (labeled $I$ ) and outputs (labeled $O$ ) are laid out literally according to the eight directions in space. In (c), the inputs are placed such at their location along $x$ determines whether they represent a corresponding direction. Both arrangements create a geometric relationship between each input and its corresponding output. In this way, it is possible the give evolution a significant advantage from the start. . . . .	29
3.6	Equivalent Connectivity Concepts at Different Substrate Resolutions. Two connectivity concepts are depicted that were evolved through interactive evolution. The CPPN that generates the first concept at $5 \times 5$ (a) <i>and</i> $7 \times 7$ (b) is shown in (c). The CPPN in (f) similarly generates the second concept at <i>both resolutions</i> (d) and (e). This illustration demonstrates that CPPNs represent a mathematical concept rather than a single structure. Thus, the same CPPN can produce patterns with the same underlying concept at different substrate resolutions (i.e. different node densities). <i>CPPN activation functions in this dissertation are denoted by <math>G</math> for Gaussian, <math>S</math> for sigmoid, <math>Si</math> for sine, <math>A</math> for absolute value, and <math>L</math> for linear.</i> . . . . .	31

4.1	HyperNEAT Performance. The performance of both sensor layouts at resolution eight with and without biasing is shown in (a). The difference in speeds when different methods are scaled from 8 to 32 is shown in (b). Graphs (c) and (d) show the speeds of the two sensor placement schemes at all resolutions with and without biasing, respectively. The conclusion is that HyperNEAT learns to exploit the placement geometry. . . . .	40
4.2	Repeated patterns in solutions. The CPPNs in (a) represent parallel (top row) and concentric (bottom row) substrates that solve the task. The connectivity patterns in (b) and (c) are outgoing <i>motifs</i> , each from a single input node in the substrate. These images show that the same motif is repeated in different locations. . . . .	44
4.3	Perfect and Imperfect Scaling. The parallel (top row) and concentric (bottom row) $8 \times 8$ motifs in (a) scale perfectly to the $16 \times 16$ motifs in (b). However, because of artifacts at higher resolution, not all motifs scale perfectly, as shown between (c) and (d), in which missing connections are identified by arrows. While the motif is mostly intact, slight imperfections of this type are common during scaling in this task. . . . .	44

4.4	Hypercube Cross-Sections. Typical (a and c) and atypical (b and d) two-dimensional cross-sections of hypercubes generated by CPPNs are depicted for working concentric (a and b) and parallel (c and d) substrate configurations. Each cross-section represents the infinite-resolution outgoing connectivity pattern originating from the location of the “X.” When substrates are generated in practice, these cross-sections are sampled at the substrate resolution. Thus, much of the detail in the patterns is discarded for low-resolution substrates, though it may reemerge when scaling to higher resolution.	46
4.5	Evolution Discovers Regularities. Early-generation connective CPPNs typically produce simple connectivity patterns (a). Eventually, HyperNEAT begins to exploit regularities (b), though they may not be the most fundamental ones. Finally, HyperNEAT discovers the fundamental regularity that underlies the task for the given substrate configuration (c). Thus, instead of optimizing individual connection weights, evolution is a process of gradually discovering and exploiting holistic regularities.	47

5.1 Multiagent HyperNEAT. This figure depicts the CPPNs and substrates that encode multiple agents with HyperNEAT. The CPPN in (a) generates a single controller for a single agent or a homogeneous team of agents. The single controller substrate that is queried by this CPPN is shown in (c). In contrast, the CPPN in (b) encodes a heterogeneous team by sampling the substrate in (d), which is made up of the single substrate (c) copied five times, but repeated across the substrate. The  $r(x)$  nodes above  $x_1$  and  $x_2$  in the initial heterogeneous CPPN (b) repeat the  $x$  coordinate frame (e), duplicating it for each agent (f) while also maintaining a global coordinate system through  $x_1$  and  $x_2$ . In this way, the CPPN can create patterns across both the agents' bodies and the team as a whole. If the same point is sampled within any two agents in (e),  $r(x)$  will return the same value (though  $x$  will not), giving agents on the team their own coordinate frame. Note that CPPNs depicted in (a) and (b) increase in complexity over evolution through the NEAT algorithm. 53

5.2	Heterogeneous seeding. From a CPPN that generates a successful single agent, multiagent HyperNEAT can generate a team of agents. To allow the CPPN to differentiate between team members, the CPPN at left is modified. The CPPN is altered so that the $x_1$ and $x_2$ inputs feed into two new nodes (shown at right). All connections that originally projected from the inputs are changed to project from these new nodes (darkened) instead. This change maintains the coordinate frame of a single agent in the $r(x)$ nodes, while the $x_1$ and $x_2$ inputs now denote the coordinate frame of the <i>team</i> , allowing the CPPN to generate patterns relevant to both. . . . .	57
6.1	Single Predator Substrate Configuration. An autonomous robot (a) is equipped with five sensors, spanning a $180^\circ$ arc in front of it and labeled 1 through 5 from left to right. The substrate that controls the robot (b) is arranged such that the placement of inputs in the ANN corresponds to the physical locations of the sensors on the robot (e.g. the leftmost sensor corresponds to the leftmost input). Similarly, the outputs of the network are related to their effects on the agent and correspond to the sensors (e.g. the left turn output is on the left side of the network and above the leftmost sensor input). Such placement allows the CPPN to easily generate connectivity patterns that respect the geometry of the problem, such as left-right symmetry. . . . .	60

6.2	Prey Formations. The prey (upper agents) can be arranged in three formations. The predators (lower agents) are always placed in the same evenly-spaced line below the prey. Each formation presents a unique challenge. . . .	61
6.3	Comparing Performance of Different Training Methods. The performance of each training method on the three formations is shown, each averaged over twenty runs. In all cases heterogeneous teams significantly outperform homogeneous teams and seeded teams outperform unseeded. . . . .	66
6.4	Generalization Performance. Average performance is shown for each approach on the nine variants of each prey formation, averaged over twenty runs. Heterogeneous methods generalize significantly better than homogeneous (at least $p < 0.01$ in all cases except unseeded heterogeneous versus seeded homogeneous on triangle) and seeding produces significantly better performance than starting from scratch (at least $p < 0.05$ in all cases). The main conclusions is that both heterogeneity and seeding afford significant advantage in generalizing in this domain. . . . .	68
6.5	Corralling Example. The predators begin by crossing to the opposite side of the prey from which they started (a), causing the prey to retreat inward. The predators then repeatedly circle the prey (b) who continue moving inward away from the predators until they are compacted enough that the predators can easily surround and capture them (c). . . . .	70

7.1	Initial Concept for Heterogeneous Scaling. Because multiagent HyperNEAT represents teams as a pattern of policies, it is possible in principle to interpolate new policies in the policy geometry for additional agents by sampling new points on the substrate. The substrate remains the same size and the additional agents are squeezed horizontally so that the new number of agents fit. Additionally, the $r(x)$ function is altered slightly so that it repeats the correct number of times. However, as explained in Section 7.1.1, this approach to interpolation can still be improved. . . . .	75
7.2	Potential Problem with Scaling. The substrate based on the $r(x)$ function has a fixed amount of space that is evenly divided among all the agents that are on the team. When additional agents are added, the agents are compressed horizontally to accommodate the new number, resulting in parts of the space that used to compose one agent now containing parts of other agents. This problem is illustrated by observing the agents' borders at team sizes five (a) and seven (b). If the pattern of policies is based heavily on the global $x$ coordinate, new policies may not correctly interpolate because of this conflation.	76

7.3	Stacked Substrate Heterogeneous Scaling. The <i>stacked substrate</i> places the ANN for each agent at a coordinate on the $z$ -axis, effectively making a stack of two-dimensional substrates. It is scaled by inserting new two-dimensional substrate slices along the $z$ -axis. Because each new slice exists at a fixed $z$ coordinate, adding new agents does not affect existing agents. This representation is thus more amenable to scaling than the divided approach (figure 7.2).	77
7.4	Stacked Substrate Seeding. The stacked substrate can also be seeded with a high-performing single-agent policy. To do so, the original CPPN (left) is given a new $z$ input that determines which agent is being sampled. This method preserves the original seed pattern, but again allows multiagent HyperNEAT to create a pattern of policies relevant to the team's policy geometry, which varies along $z$ .	78
7.5	Training Performance of Different Substrates. The training performance of the various teams is shown for each of the three formation shapes, averaged over twenty runs. In all cases, the seeded stacked substrate teams learned the best solutions the fastest. Additionally, heterogeneous teams consistently outperformed pure homogeneous teams and seeded teams consistently outperformed their unseeded counterparts.	80



7.6	Scaling Success Rates of Different Substrates (lower is better). The number of trials in which all prey were captured is shown for each of the three formation shapes, summed over twenty runs, with two trials per run. Success rates were determined by the averaging the performance of the team from each run with the best performance on all team sizes over all twenty runs. Scaling performance generally mirrors training performance: Seeded stacked is the best overall, heterogeneous is better than pure homogeneous, and seeded is better than unseeded. . . . .	84
7.7	Scaling Performance of Different Substrates (lower is better). The average amount of time taken out of 2,000, summed over both formations, is shown for each team composition averaged over 20 runs on various team sizes on the three formations. A score of 4,000 means no team was able to complete that size. Scaling performance is determined by averaging the performance of the team from each run with the best performance on all team sizes over all twenty runs. Again, seeded stacked is the best overall, heterogeneous is better than pure homogeneous, and seeded is better than unseeded. . . . .	85
7.8	Room Clearing Domain. The agents in the room clearing domain (a) are represented by the small blue circle. They have 11 rangefinder sensors denoted by the rays emanating from the agent and have a visual range depicted by the large yellow circle. The agents must enter a room and spread out so that their combined visual ranges cover as much area as possible (b). . . . .	87

7.9	Room Clearing Results. In the room-clearing domain performance (defined as sum of the number of grid squares covered by the team during the evaluation period) is averaged over twenty runs. Heterogeneous teams learned the best solutions and learned them more quickly than homogeneous. Additionally, heterogeneous teams were better able to scale at all teams sizes. In fact homogeneous teams actually begin to lose performance as team size grows due to inefficiencies with their strategies. . . . .	90
8.1	Prey Formations. The prey (red circles) are arranged in either a line (a) or an L (b) formation, thereby testing both symmetric and asymmetric scenarios. This figure depicts both formations with eight predators (blue squares), although the number of predators and prey can change. The predators are always placed in an evenly-spaced line below the prey. . . . .	97
8.2	Line Pre-Training Scaling. The performance of each method for pre-training scaling on the line formation is shown. In almost all cases multiagent HyperNEAT teams are able to find better solutions more quickly than SARSA( $\lambda$ ). After 16 agents, SARSA( $\lambda$ ) can no longer solve the task, while HyperNEAT is still able to solve it up to 256 agents. Results are averaged over ten runs. .	102

8.3	L Pre-Training Scaling. The L formation was more difficult than the line due to the exploration necessary to solve the problem. However, multiagent HyperNEAT was able to find solutions consistently for all tested sizes. SARSA( $\lambda$ ) could only find solutions at the smallest size of four, and seeded SARSA( $\lambda$ ) was unable to find any solutions. Note that because the distance between the furthest prey and the predators doubles each time the size increases, it is also reasonable for the time to capture the prey to double as well. Results again are averaged over ten runs. . . . .	103
8.4	Line Post-Training Scaling 2-16 (higher is better). The number of successful scalings (both up and down) out of ten runs for different team sizes is shown. For each graph the training size is shaded black. While HyperNEAT is able to find scalable solutions at all sizes without further learning, SARSA( $\lambda$ ) can only scale from two-agent solutions (up to at most 16). . . . .	107

8.5	Line Post-Training Scaling 32-256 (higher is better). Again, the number of successful scalings (both up and down) out of ten runs is shown, but for the larger team sizes that SARSA( $\lambda$ ) could not solve during training. For each graph the training size is shaded black. Scaling up at larger sizes becomes more difficult because of the different strategies required for different sizes and the sheer magnitude of the increase in agents (e.g. 256 to 512 versus four to eight). Nevertheless, multiagent HyperNEAT is able to find solutions that scale, even up to 1,024 agents, a size that could not be learned when specifically trained to do so. . . . .	108
8.6	L Post-Training Scaling (higher is better). Successful scalings (out of 10) are shown on these graphs for different team sizes. The training size that is scaled from is shown in black. HyperNEAT was generally able to find scalable solutions for all sizes, while SARSA( $\lambda$ ) could not scale at all. . . . .	109

8.7	Typical Strategies for 16 Predators. At size 16 predators (blue squares at the top of the pictures) SARSA( $\lambda$ )-trained teams typically try to surround the prey (green squares closer to the bottom) by employing the predators on the edges to move to behind the prey (a) and push them towards the center (b) where they can be captured (c). Multiagent HyperNEAT teams also learn this strategy, but eventually develop a more complex version wherein the predators divide the prey into two groups (d) and then surround (e) and capture them independently (f). The multiagent HyperNEAT result is more efficient because more prey are captured in parallel, and it is more scalable because eventually the size of the prey lines becomes too large to traverse within the time limit. . . . .	112
8.8	Multiagent HyperNEAT Strategies for Large Teams. For larger teams such as the team of 64 predators in this figure, multiagent HyperNEAT typically continues the strategy of dividing the prey into groups (a), surrounding (b), and capturing them (c). The difference from smaller sizes is that they are divided into more and more groups so that larger numbers of prey can still be efficiently captured by the predators. . . . .	113

8.9	Multiagent HyperNEAT 256 Agent Behaviors. At the very largest trained size of 256, multiagent HyperNEAT sometimes found a strategy wherein every other predator does nothing and the remaining predators move forward (a). This strategy causes the prey to bounce between the charging predators (b) until they are all eventually captured (c). Such a tactic is extremely efficient at this problem size and requires strict cooperation of almost every agent to be successful. . . . .	114
9.1	Multiagent HyperNEAT with Situational Policy Geometry. In standard multiagent HyperNEAT, a substrate (a) is changed into a multiagent substrate by adding an additional input(s) to the CPPN (b) that represents the policy geometry of the team. Situational policy geometry is handled similarly: The CPPN (c) has an additional input $S$ that describes the location in the situational policy geometry for a given agent's controller (d), which is formalized in the new $S$ -axis in the situational team substrate (e). Thus the network stack to the left along $S$ is team policies for one situation while that on the right is policies for a different situation. . . . .	124

9.2	Khepera III with Korebot II. The Khepera III mobile robots (a) in these experiments come equipped with a Korebot II extension that runs an embedded Linux operating system and allows the robots to receive broadcast communications over a wireless network. Although the Khepera III has many sensors available, only the front six infrared rangefinders (b) are utilized in these experiments. . . . .	126
9.3	Real-World Environment. The real environments with which the robots interact are constructed out of red bricks on a carpet with the same dimensions used in the simulator. The plus (a) is the environment the robots are trained on, and the asymmetric plus (b) tests the generality of the learned policies. In both cases robots are placed 30cm apart in the open branch and then sent a signal to begin patrolling. Individual robots can then be called back by the experimenter by broadcasting a command to them. . . . .	129

## LIST OF TABLES

A.1	Parameter Settings in Experiments . . . . .	149
-----	---	-----



# CHAPTER 1

## INTRODUCTION

While there are many problems that can be solved with a single agent, many others are either too difficult or too time consuming to be solved by one individual [SSS01, RW03, Tan97]. This problem is the focus of the field of cooperative multiagent learning, in which teams of agents are trained to perform tasks with a common goal. Currently, existing approaches such as cooperative coevolutionary algorithms (CCEAs; [PL05, PWL03, PDG95, PMS01, FP00]) and multiagent reinforcement learning (MARL; [HW98, CB98, SH02, BV02]) generally treat the multiagent problem as many single-agent learning problems that consider the states of other agents. This dissertation introduces the novel approach of representing and training teams based on discovering how their roles relate to each other

The problem of deciding who on a team should perform which tasks (i.e. role allocation) is complex and well studied [CW10]. However, an important observation of real-life teams is that the behaviors and policies of team members tend to be dictated by their canonical position within the team (e.g. at the start of a sporting match). The result is that the policies of agents on a team are often conceptually distributed in a spatial pattern according to their positions. In other words, the team has a *policy geometry* that dictates how each member should behave based on their location within the team. For example, in a soccer

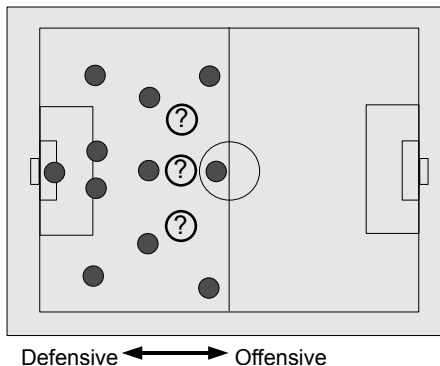


Figure 1.1: Role as a Function of Geometry. In the depicted soccer team, the defensive to offensive dimension of variation spans from left to right. The question marks represent a hypothetical new line of players, whose roles can be inferred from their positions. The main conclusion is that policies are distributed in a *pattern* in space.

(i.e. football) team (figure 1.1), the positions closest to the goal are defensive and become incrementally more offensive the farther they are from the goal. Also importantly, even as policies vary across space, agents tend to *share* common skills; in the soccer example, all players know how to pass and kick. Thus, rather than learning the policies of individual agents, an intriguing possibility for multiagent learning is to learn how the pattern of policies relates to the geometry of the team and *derive* the policies of individual agents based on that relationship.

To see the benefits of this perspective, consider that in traditional approaches, the complexity of the multiagent learning problem increases with the size of the team because the number of possible states grows with the number of agents, which limits the size of teams that can be trained [BBS08, PL05]. However, representing multiagent teams as a *pattern of policies* rather than as individual agents eliminates this problem because all that needs to be learned and represented is the pattern. Even more interesting is that such a pattern

effectively represents up to an infinite number of multiagent policies that can be sampled from the *policy geometry* as needed; thus multiagent teams represented in this way have no *a priori* bound on their team size.

While homogeneous and heterogeneous learning are usually separated conceptually in discussions of multiagent learning [WKF09, PL05, SV00], another important insight is that heterogeneity can alternatively be viewed as a *continuum* that begins at pure homogeneity and ends at radical differentiation. For example, in this view, it is possible for a team to be nearly homogeneous but not completely. In such a team, to a large extent, many skills and behaviors are shared among agents on the team even though individuals possess their own unique tendencies, suggesting the possibility to avoid redundantly discovering and representing common skills. Such intermediate configurations between pure homogeneity and extreme heterogeneity are common in human teams. For example, in soccer, players *share* many fundamental abilities, from low-level universal human capacities such as visual recognition to high-level skills such as passing and dribbling the ball. In fact, it is difficult to think of a kind of team that is purely heterogeneous (i.e. no team members share any traits or skills whatsoever).

This view of heterogeneity as extending from pure homogeneity exposes a potential problem with approaches that treat heterogeneous agents as separately learned entities. This *problem of reinvention* is that many of the shared skills that often should dominate the policies of every agent in fact must be reinvented separately for each agent. Thus a more principled approach should be able to represent the commonalities as a regularity that only

needs to be discovered and encoded once. The approach in this dissertation shows how such team encoding is possible through learning policy geometry. Interestingly, once regularities can be encoded, it becomes possible to scale heterogeneous teams by interpolating new policies based on encoded regularities. In this way, the idea of a continuum of heterogeneity facilitates not only learning but scaling.

Thus this dissertation introduces a method that both addresses the problem of reinvention and provides a way to overcome the computational obstacles to scaling the size of multi-agent teams by representing them as patterns of policies rather than as individual agents. To implement this idea, the dissertation first introduces *hypercube-based neuroevolution of augmenting topologies* (HyperNEAT), an approach to evolving artificial neural networks (ANNs) that was co-invented by myself, Jason Gauci, and Kenneth Stanley and has demonstrated success in single-agent control and in encoding scalable neural networks [SDG09]. HyperNEAT is then extended to encode *patterns of ANNs distributed across space* with a single genome. The spatial distribution of ANNs matches with the locations of agents on the team, thereby allowing HyperNEAT to learn a *pattern of policies* (i.e. the policy geometry), all generated from the same genome. In this way, HyperNEAT can reuse critical information to conquer the problem of reinvention. This new method is called *multiagent HyperNEAT*. Additionally, because teams are represented as patterns and not single agents, multiagent HyperNEAT can derive the policies of previously non-existent agents, allowing teams to scale in size *without further learning*, a novel and powerful capability.

## 1.1 Contributions

The research hypothesis of this dissertation is that a multiagent learning method that exploits policy geometry and the continuum of heterogeneity, such as multiagent HyperNEAT, can create large, robust, and scalable multiagent teams. This hypothesis is supported by the following contributions:

1. The HyperNEAT indirect encoding is introduced, which was co-invented by myself, Jason Gauci, and Kenneth O. Stanley.
2. Multiagent HyperNEAT extends HyperNEAT to allow teams of heterogeneous agents to be encoded by a single genome and exploit the *continuum of heterogeneity*, thereby overcoming the *problem of reinvention*.
3. A method to scale teams trained by multiagent HyperNEAT to significantly larger sizes makes possible scaling up by several orders of magnitude.
4. A comprehensive study compares multiagent HyperNEAT and a traditional multiagent training method, SARSA( $\lambda$ ), focusing on both training teams of varying sizes and scaling trained policies to new sizes.
5. A method is introduced to train multiple policies for each individual agent on a team, among which the agent can switch depending on its state, allowing each agent to easily perform multiple tasks.

6. Teams trained with multiagent HyperNEAT in a simulator are transferred to a teams of real Khepera III robots to perform a real-world patrol task.

## 1.2 Outline

The next chapter begins with relevant background in cooperative multiagent learning as well as neuroevolution and generative systems, the backbones of the multiagent HyperNEAT method. Chapter 3 introduces the original HyperNEAT method, which produces neural networks as a function of their geometry. Next, a single-agent control experiment is discussed in Chapter 4 that tests the HyperNEAT method and serves as a stepping stone to multiagent control, which is the focus of Chapter 5. Chapter 6 then tests the multiagent HyperNEAT method in a predator-prey experiment and Chapter 7 takes the predator-prey domain further by introducing a new team representation and demonstrating the ability for teams to scale without additional training. To establish multiagent HyperNEAT as a competitive method in the field, Chapter 8 compares it to an established reinforcement learning technique called multiagent *SARSA*( $\lambda$ ). Teams trained with an extended version of multiagent HyperNEAT that can switch among tasks are then transferred to real robots in a real-world patrol task in Chapter 9. Finally, Chapter 10 discusses the implications of the above experiments and the dissertation as a whole.

## **CHAPTER 2**

### **BACKGROUND**

This chapter provides background on several areas relevant to multiagent HyperNEAT, including cooperative multiagent learning, neuroevolution and generative and developmental systems.

#### **2.1 Cooperative Multiagent Learning**

Multiagent systems confront a broad range of domains, from predator-prey confrontations to military scenarios, creating the opportunity for real-world applications. In cooperative multiagent learning, which is reviewed in this section, agents are trained to work together to accomplish a task, usually by one of several alternative methods.

### *2.1.1 Traditional Approaches*

There are two major classes of approaches to multiagent learning: multiagent reinforcement learning (MARL; [SS01, CS07, ICM94, SKM00]) and cooperative coevolutionary algorithms (CCEAs; [PL05, PD94, FP00]). While these approaches are mainly the focus of separate communities, Panait et. al [PTL08] noted recently that they share a significant common theoretical foundation. One key commonality is that they break the learning problem into separate roles that are semi-independent and thereby learned separately through interaction with each other. Although this idea of separating multiagent problems into parts is popular, it does create challenges for certain desirable objectives such as scaling to larger team sizes, a focus of this dissertation. The problem is that when individual roles are learned separately, there is no representation of how roles relate to the team structure and typically the only way to incorporate new agents is to create duplicates of existing agents, which precludes the possibility of introducing new roles automatically. Nevertheless, these approaches have produced significant insight into multiagent learning, and are reviewed in this section.

Multiagent reinforcement learning (MARL) encompasses several specific techniques based on off-policy and on-policy temporal difference learning [HW98, CB98, SH02, BV02]. The basic principle that unifies MARL techniques is to identify and reward promising cooperative states and actions among a team of agents [BSB05, PL05]. For example, if a group of predator agents is tasked with capturing a prey, agents are rewarded at least in part when the prey is captured. The reward is a signal that their recent actions were useful and should be repeated



with greater probability. Conversely, actions that are not rewarded or that are punished are less likely to be repeated. Rewards can be given globally (to the entire team) or locally (to single agents or groups of agents) depending on the specific method chosen.

One reason that MARL is attractive is because it is possible to prove convergence to Nash equilibria in some situations [HW03, SKM00]. Yet such guarantees rely on having complete or at least significantly large amounts of information about the world state, and lacking such information is often what makes multiagent domains challenging [SPG04]. Even in scenarios with sufficient state information, the number of states grows with the number of agents, making the state space expensive to represent. Additionally, there is the credit assignment problem [Mat97], wherein it is not always possible for the individual agents to associate the rewards with the correct actions. Thus experimenters must carefully identify the correct reward states manually to minimize this uncertainty. Nevertheless, MARL has provided several successes including intrusion detection [SK08] and trading strategies [KUP03].

The other major approach, CCEAs, is an established evolutionary method for training teams of agents that must work together [PL05, PWL03, PMS01, FP00]. The main idea is to maintain one or more populations of candidate agents, evaluate them in groups, and guide the creation of new candidate solutions based on their joint performance. An important distinction among coevolutionary methods is the population model chosen, which can range from a single population from which all team members are drawn [BH97, PSG98] to separate populations for each agent on the team [PMS01]. Recent work by Panait et. al

[PLH06] has shown that CCEA performance is enhanced by keeping an archive of informative collaborators, an idea that is also relevant to MARL.

In cooperative coevolution, agents are explicitly rewarded for their cooperative abilities and multiple candidate solutions may be evaluated in the same population, potentially leading to more robust solutions. However, depending on the population model employed, trade-offs between specialization and skill sharing must be made. In the single population model, genetic information is easily spread among the entire population; thus if an agent finds a good general strategy other agents can potentially adopt it. However, with only one population it is difficult for agents to specialize to specific roles [Wie04]. In contrast, multiple populations encourage specialization, yet the separate populations must reinvent basic, useful skills. Additionally, by evaluating agents with many different teams, depending on the evaluation scheme there is a risk of encouraging too much generalization [PLW06], which means the resulting team may not be the best possible. Another significant problem is that adding more agents to the team results in a significant increase in computational complexity; the need for more sampling and potentially more populations and can degrade the ability to optimize performance of the final team [Mic03].

Both CCEAs and MARL face the *problem of reinvention*. That is, because agents are treated as separate subproblems they must usually separately discover and represent all aspects of the solution, even though there may be a high degree of overlapping information among the policies of each agent. CCEAs commonly separate agents into different populations, creating strict divisions among agents, and in MARL methods, each agent learns

a separate reward function based upon individual experiences. There have been attempts to address the problem of reinvention such as introducing existing agents that “train” new agents [Tan97, PB99] or implementing specially-designed genetic operators [HS96]. However, an intriguing alternative is to exploit the *continuum of heterogeneity*, which means distributing shared skills optimally among the agents and only representing such skills once. At the same time, unique abilities could be isolated and assigned appropriately. The approach in this dissertation addresses the problem of reinvention by allowing the learning algorithm to find the right point on the continuum of heterogeneity for a given team.

### ***2.1.2 Alternative Techniques***

Breaking the team into separate parts is not the only way to distribute policies. This section reviews several alternative approaches.

One such approach is to optimize a single, monolithic controller that takes inputs from all the agents and outputs the actions of all the agents. Such consolidation allows information sharing but generally increases the dimensionality of the search significantly, while ignoring separability [YM07]. Such a model also assumes the existence of global, instantaneous communication, which is infeasible in most real-world scenarios. A related approach is to directly encode several disconnected policies in a single monolithic description [Mic03, Bon00, LS96], which gains separability at the expense of information sharing. In both cases, adding more

agents to the teams causes large increases in the search space, especially in the single controller case.

One solution to reduce dimensionality in either case without combining multiple individuals is to assign the same homogeneous control system to each agent [BM03, BTB07]. If all the agents are controlled by separate instantiations of a single controller, then it is only necessary to discover that one policy, and the problem of sharing discoveries disappears. This approach has produced significant results in swarm robotics, such as a team of four connected robots that exhibit coordinated trajectories [BTB07]. However, Yong and Miikkulainen [YM07] show that in predator-prey tasks with three predators and one prey, heterogeneous teams learn more effective strategies than homogeneous ones. Thus, while a homogeneous team may be easier to evolve and scale, there are limits to what it can do [WKF09]. Pure homogeneity is only a single point on the continuum.

In summary, there are many challenges that face multiagent learning, and choosing one method over another generally leads to trade-offs among several competing factors. However, in almost all cases, adding more agents to the team greatly impacts the efficiency of search and, as a result, the overall quality of the resulting multiagent team. Additionally, adjustments to team size must be made *before* learning; thus any post-hoc adjustments require at least some retraining.

The next section reviews the NEAT method, the foundation for HyperNEAT and multiagent HyperNEAT (the approaches introduced in this dissertation), which address multiagent learning in a completely different way than the techniques reviewed in this section.

## 2.2 NeuroEvolution of Augmenting Topologies (NEAT)

The NEAT method was originally developed to evolve ANNs to solve difficult control and sequential decision tasks [SM02b, SM04a, SBM05b]. In this dissertation, it is significantly extended to evolve the representation of teams of agents. Nevertheless, the basic principles of NEAT, reviewed in this section, still supply the foundation of the approach.

Traditionally, ANNs evolved by NEAT control agents that select actions based on their sensory inputs. NEAT is unlike many previous methods that evolved neural networks, i.e. *neuroevolution* methods, which historically evolved either fixed-topology networks [GM99, SF95], or arbitrary random-topology networks [ASP93, GWP96, Yao99]. Instead, NEAT begins evolution with a population of small, simple networks and increases the complexity of the network topology into diverse species over generations, leading to increasingly sophisticated behavior. A similar process of gradually adding new genes has been confirmed in natural evolution [Mar99, WHR87] and shown to improve adaptation in a few prior evolutionary [Alt94] and neuroevolutionary [Har93] approaches. However, a key feature that distinguishes NEAT from prior work in evolving increasingly complex structures is its unique approach to maintaining a healthy diversity of structures of different complexity simultaneously, as this section reviews. This approach has proven effective in a wide variety of domains [Aal09, SBM05a, SKM05, TWS06, HRS08, RVH09, SM02a, TOL08, KSM06, SM04b, SBD08, HS09, HGS09b, HGS09a, SM04a]. Complete descriptions of the NEAT method, including

experiments confirming the contributions of its components, are available in Stanley and Miikkulainen [SM02b, SM04a] and Stanley et. al [SBM05b].

The NEAT method is based on three key ideas. First, to allow network structures to increase in complexity over generations, a method is needed to keep track of which gene is which. Otherwise, it is not clear in later generations which individual is compatible with which in a population of diverse structures, or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique *historical marking* to every new piece of network structure that appears through a structural mutation. The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited during crossover unchanged, and allow NEAT to perform crossover among diverse topologies without the need for expensive topological analysis.

Second, NEAT speciates the population so that individuals compete primarily within their own niches instead of with the population at large. Because adding new structure is often initially disadvantageous, this separation means that unique topological innovations are protected and therefore have the opportunity to optimize their structure without direct competition from other niches in the population. NEAT uses the historical markings on genes to determine to which species different individuals belong.

Third, many approaches that evolve network topologies and weights begin evolution with a population of random topologies [GWP96, Yao99]. In contrast, NEAT begins with a uniform population of simple networks with no hidden nodes, differing only in their initial

random weights. Because of speciation, novel topologies gradually accumulate over evolution, thereby allowing diverse and complex phenotype topologies to be represented. No limit is placed on the size to which topologies can grow. New nodes and connections are introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. In effect, then, NEAT searches for a compact, appropriate topology by incrementally adding complexity to existing structure.

The next section reviews the field of generative and developmental systems (GDS) with a focus on compositional pattern producing networks (CPPNs), which combine with NEAT to make the HyperNEAT method.

### 2.3 Generative and Developmental Systems

A key similarity among many neuroevolution methods, including NEAT, is that they employ a *direct encoding*, that is, each part of the solution’s representation maps to a single piece of structure in the final solution [SM02b, GM97, Yao99, FDM08]. Yet direct encodings impose the significant disadvantage that even when different parts of the solution are similar, they must be encoded and therefore discovered separately. This challenge is related to the *problem of reinvention* in multiagent systems: After all, if individual team members are encoded by separate genetic code, even if a component of their capabilities is shared, the search algorithm has no way to exploit such a regularity. Thus this dissertation exploits the power of *indirect*

encoding instead, which means that the description of the solution is compressed such that information can be reused, allowing the final solution to contain more components than the description itself. Indirect encodings are often motivated by *development* in biology, in which the genotype maps to the phenotype indirectly through a process of growth [BK99, Lin74, SM03]. Indirect encodings are powerful because they allow solutions to be represented as a pattern of policy parameters, rather than requiring each parameter to be represented individually. This capability is the focus of the field called *generative and developmental systems* [BK99, Bon02, Egg97a, HP02, Lin74, Mil04, Sim94a, Sta07, SM03]. This section examines the field in greater depth and focuses on a particular encoding called *compositional pattern producing networks*.

### 2.3.1 *Compositional Pattern Producing Networks (CPPNs)*

In biological genetic encoding the mapping between genotype and phenotype is *indirect*, which means that the phenotype typically contains orders of magnitude more structural components than the genotype contains genes. For example, a human genome of 30,000 genes (about three billion amino acids) encodes a human brain with 100 trillion connections [Del95, KSJ91, DSG98]. Thus, the only way to discover structures with trillions of parts may be through a mapping between genotype and phenotype that translates few dimensions into many, i.e. through an indirect encoding. Because phenotypic structures often occur



in repeating patterns, each time a pattern repeats, the same gene group can provide the specification. The numerous left/right symmetries of vertebrates [Raf96, pages 302-303], the receptive fields in the visual cortex [GW92, HW65], and fingers and toes are examples of repeating patterns in biology.

A most promising area of research in indirect encoding is *developmental encoding*, which is motivated from biology [Ang95, BK99, HP02, SM03]. Development facilitates reusing genes because the same gene can be activated at any location and any time during the development process.

This observation has inspired an active field of research in artificial developmental encodings [Ang95, BK93, BK99, Bon02, Daw86, DB96, Egg97b, Fed04b, Fed04a, Gru94, HP02, Jak95, KR01, Lin68, Lin74, Mil04, MSR91, PL90, Sim94b, SM03, Tur52]. The aim is to find the right abstraction of natural development for a computer running an evolutionary algorithm, so that evolutionary computation (EC) can begin to discover complexity on a natural scale. Prior abstractions range from low-level cell chemistry simulations to high-level grammatical rewrite systems [SM03].

CPPNs are a novel abstraction of development that can represent sophisticated repeating patterns in Cartesian space [Sta06, Sta07]. Unlike most generative and developmental encodings, CPPNs do not require an explicit simulation of growth or local interaction, yet still realize their essential functions. This section reviews CPPNs, which will be augmented in the HyperNEAT method (Chapter 3) to represent connectivity patterns and ANNs.

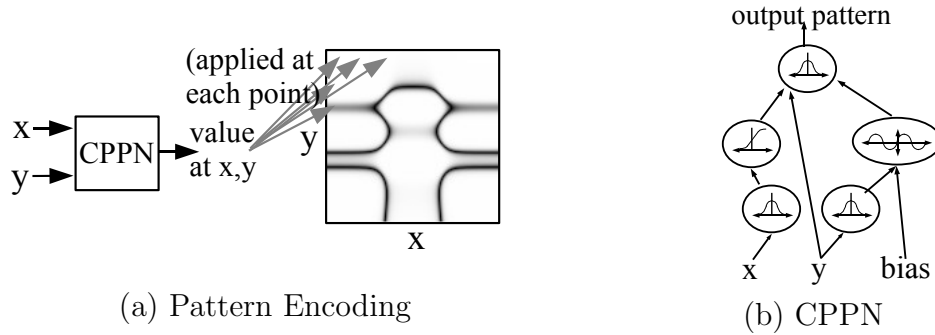


Figure 2.1: CPPN Encoding. (a) The CPPN takes arguments  $x$  and  $y$ , which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of the CPPN, the result is a spatial pattern, which can be viewed as a phenotype whose genotype is the CPPN. (b) Internally, the CPPN is a graph that determines which functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. The CPPN in (b) actually produces the pattern in (a)

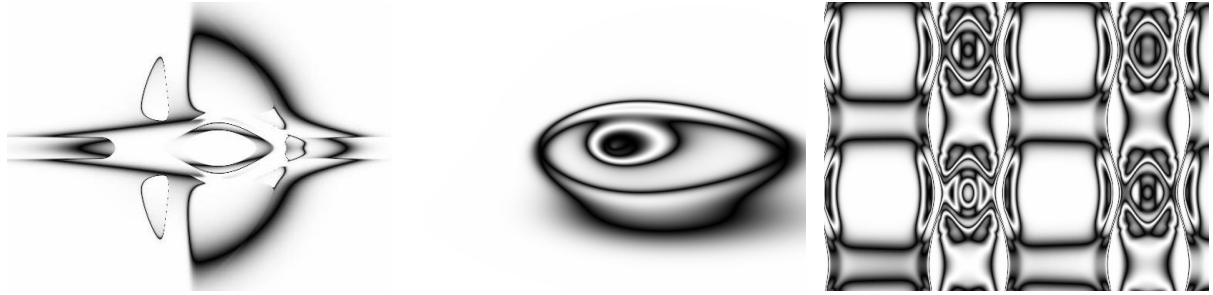
Consider the phenotype as a function of  $n$  dimensions, where  $n$  is the number of dimensions in physical space. For each coordinate in that space, its level of expression is an output of the function that encodes the phenotype. Figure 2.1a shows how a two-dimensional phenotype can be generated by a function of two parameters.

Stanley [Sta06, Sta07] showed how simple canonical functions can be composed to create an overall network that produces complex regularities and symmetries. Each component function creates a novel geometric *coordinate frame* within which other functions can reside. The main idea is that these simple canonical functions are abstractions of specific events in development such as establishing bilateral symmetry (e.g. with a symmetric function such as Gaussian) or the division of the body into discrete segments (e.g. with a periodic function such as sine). Figure 2.1b shows how such a composition can be represented by a network.

Such networks are called *compositional pattern producing networks* because they produce spatial patterns by composing basic functions. Unlike ANNs, which often contain only sigmoid functions (or sometimes Gaussian functions), CPPNs can include both types of functions and many others. Furthermore, the term *artificial neural network* would be misleading in the context of this research because ANNs were so named to establish a metaphor with a different biological phenomenon, i.e. the brain. The terminology should avoid making the implication that biological, thinking brains are in effect the same as developing embryos or genetic encodings. In this dissertation, because CPPNs are used to encode ANNs, it is especially important to differentiate these concepts.

Through interactive evolution, Stanley [Sta06, Sta07] demonstrated that CPPNs can produce spatial patterns with important geometric motifs that are expected from generative and developmental encodings and seen in nature. Among the most important such motifs are symmetry (e.g. left-right symmetries in vertebrates), imperfect symmetry (e.g. right-handedness), repetition (e.g. receptive fields in the cortex [ZBL99]), and repetition with variation (e.g. cortical columns [GC02]). Figure 2.2 shows examples of several such important motifs produced through interactive evolution of CPPNs.

It is fortuitous that CPPNs and ANNs are so similar from a structural perspective because methods designed to evolve ANNs can also evolve CPPNs. In particular, the NEAT method is a good choice for evolving CPPNs because NEAT increases the complexity of evolving networks over generations, allowing increasingly elaborate regularities to accumu-



(a) Symmetry

(b) Imperfect Symmetry

(c) Repetition with Variation

Figure 2.2: CPPN-generated Regularities. Spatial patterns exhibiting (a) bilateral symmetry, (b) imperfect symmetry, and (c) repetition with variation (notice the nexus of each repeated motif) are depicted. These patterns demonstrate that CPPNs effectively encode fundamental regularities of several different types.

late. The next chapter describes the HyperNEAT method that combines CPPNs and NEAT to generate ANNs with geometric regularities.

## CHAPTER 3

### HYPERNEAT

If CPPNs are to evolve and represent connectivity patterns, the problem is to find the best interpretation of their output to effectively describe such a structure. The two-dimensional patterns in Section 2.3.1 present a challenge: How can such spatial patterns describe connectivity? This chapter explains how spatial patterns generated by CPPNs can be mapped naturally to connectivity patterns while at the same time effectively disentangling task structure from network dimensionality in a method called HyperNEAT. This novel representation allows neurons, sensors, and effectors to exploit meaningful geometric relationships. HyperNEAT was invented by myself, Jason Gauci and Kenneth Stanley at the University of Central Florida [DS07, DS08, GS07, GS08, SDG09] and has since been used by several other researchers in a variety of domains [CPO09, CBO09, DKS09, CBP09, BKS09]. The text in this chapter is based on our 2009 journal article [SDG09]. The next section introduces the key insight behind HyperNEAT, which is to assign connectivity a geometric interpretation.

### 3.1 Mapping Spatial Patterns to Connectivity Patterns

It turns out that there is an effective mapping between spatial and connectivity patterns that can elegantly exploit geometry. The main idea is to input into the CPPN the coordinates of the *two points* that define a connection rather than inputting only the position of a single point as in Section 2.3.1. The output is interpreted as the *weight* of the connection rather than the intensity of a point. In this way, connections can be defined in terms of the locations that they connect, thereby taking into account the network’s geometry. The CPPN in effect computes a four-dimensional function  $CPPN(x_1, y_1, x_2, y_2) = w$ , where the first node is at  $(x_1, y_1)$  and the second node is at  $(x_2, y_2)$ . This formalism returns a weight for every connection between every node in a set of nodes, including recurrent connections. By convention, a connection is not expressed if the magnitude of its weight, which may be positive or negative, is below a minimal threshold  $w_{min}$ . The magnitude of weights above this threshold are scaled to be between zero and a maximum magnitude. That way, the pattern produced by the CPPN can represent any network topology (figure 3.1).

For example, consider a grid of nodes, as in figure 3.1. The nodes are assigned coordinates corresponding to their positions within the grid (labeled *substrate* in figure 3.1), where  $(0, 0)$  is the center of the grid. Assuming that these nodes and their positions are given *a priori*, a connectivity pattern among nodes in two-dimensional space is produced by a CPPN that takes any two coordinates (source and target) as input, and outputs the weight of their connection. The CPPN is queried in this way for every potential connection on the grid.

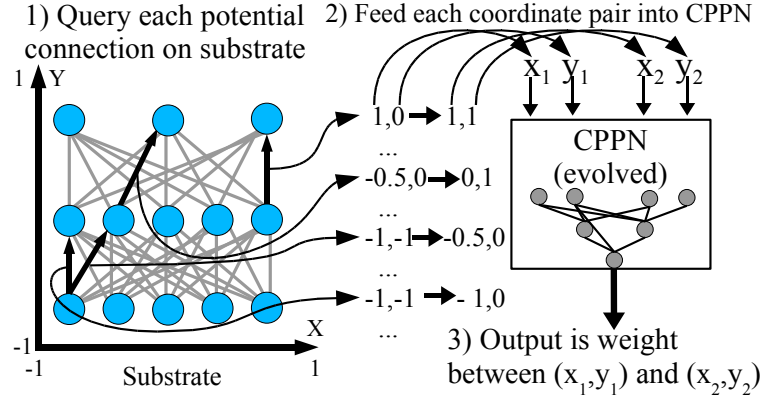


Figure 3.1: Hypercube-based Geometric Connectivity Pattern Interpretation. A collection of nodes, called the *substrate*, is assigned coordinates that range from  $-1$  to  $1$  in all dimensions. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines in the substrate depicted in the figure represent a sample of connections that are queried. (2) For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. Thus, CPPNs can produce regular patterns of connections in space.

Because the connection weights are thereby a function of the *positions* of their source and target nodes, the distribution of weights on connections throughout the grid will exhibit a pattern that is a function of the geometry of the coordinate system.

The connectivity pattern produced by a CPPN in this way is called the *substrate* so that it can be verbally distinguished from the CPPN itself, which has its own internal topology. Furthermore, in the remainder of this dissertation, CPPNs that are interpreted to produce connectivity patterns are called *connective CPPNs* while CPPNs that generate spatial patterns are called *spatial CPPNs*. This dissertation focuses on neural substrates produced by connective CPPNs.

Because the connective CPPN is a function of four dimensions, the two-dimensional connectivity pattern expressed by the CPPN is isomorphic to a spatial pattern embedded in a four-dimensional hypercube. This observation is important because it means that spatial patterns with symmetries and regularities correspond to connectivity patterns with related regularities. Thus, because CPPNs generate regular spatial patterns (Section 2.3.1), by extension they can be expected to produce connectivity patterns with corresponding regularities. The next section demonstrates this capability.

### 3.2 Producing Regular Connectivity Patterns

Simple, easily-discovered substructures in the connective CPPN produce important connective regularities in the substrate. The key difference between connectivity patterns and spatial patterns is that each discrete unit in a connectivity pattern has *two*  $x$  values and *two*  $y$  values. Thus, for example, symmetry along  $x$  can be discovered simply by applying a symmetric function (e.g. Gaussian) to  $x_1$  or  $x_2$  (figure 3.2a).

Imperfect symmetry is another important structural motif in biological brains. Connective CPPNs can produce imperfect symmetry by composing *both* symmetric functions of one axis along with an asymmetric coordinate frame such as the axis itself. In this way, the CPPN produces varying degrees of imperfect symmetry (figure 3.2b).



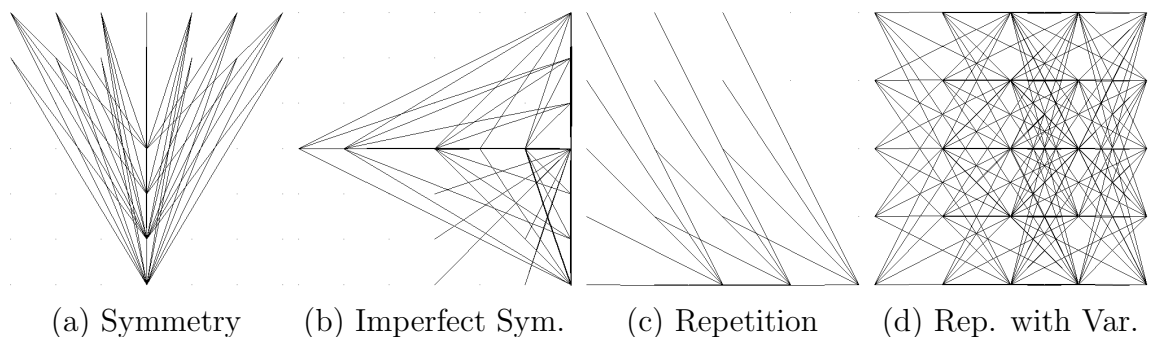


Figure 3.2: Connectivity Patterns Produced by Connective CPPNs. These patterns, produced through interactive evolution, exhibit several important connectivity motifs: (a) bilateral symmetry, (b) imperfect symmetry, (c) repetition, and (d) repetition with variation. That these fundamental motifs are compactly represented and easily produced suggests the power of this encoding.

Similarly important is repetition, particularly repetition with variation. Just as symmetric functions produce symmetry, periodic functions such as sine produce repetition (figure 3.2c). Patterns with variation are produced by composing a periodic function with a coordinate frame that does not repeat, such as the axis itself (figure 3.2d). Repetitive patterns can also be produced in connectivity as functions of invariant properties between two nodes, such as distance along one axis. Thus, symmetry, imperfect symmetry, repetition, and repetition with variation, key structural motifs in all biological brains, are compactly represented and therefore easily discovered by CPPNs.

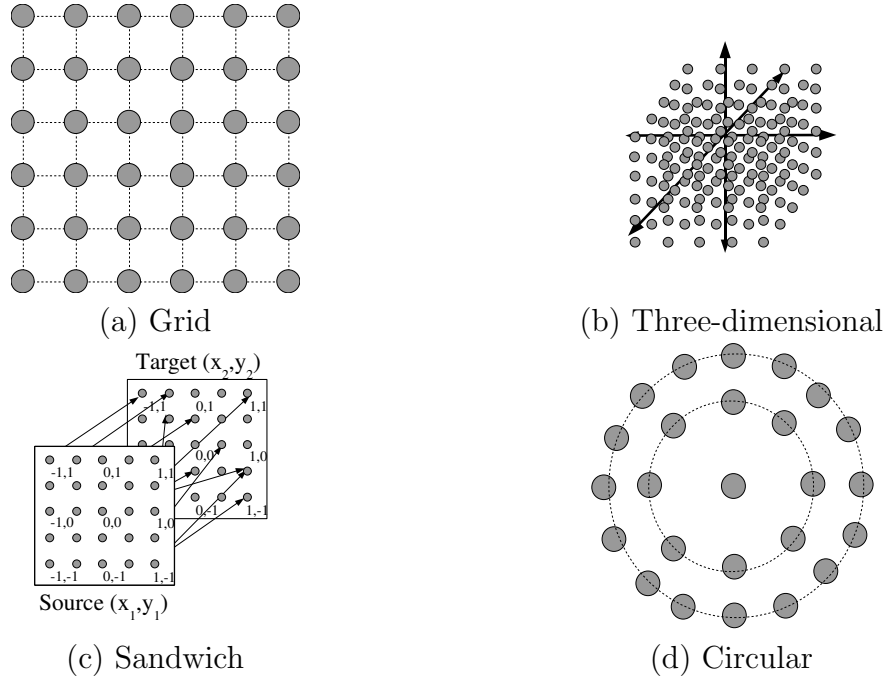


Figure 3.3: Alternative Substrate Configurations. The two-dimensional grid configuration (a), similar to the one in figure 3.1, is only one of many potential substrate configurations. This figure also shows (b) a three-dimensional configuration of nodes centered at  $(0, 0, 0)$ , (c) a “state-space sandwich” configuration in which a source sheet of neurons connects directly to a target sheet, and (d) a circular configuration. Different configurations are suited to problems with different geometric properties.

### 3.3 Substrate Configuration

The layout of the nodes that the CPPN connects in the substrate can take forms other than the planar grid (figure 3.3a) discussed thus far. Different such *substrate configurations* are likely suited to different kinds of problems.

For example, CPPNs can also produce *three-dimensional* connectivity patterns by representing spatial patterns in the *six-dimensional* hypercube  $CPPN(x_1, y_1, z_1, x_2, y_2, z_2)$  (fig-

ure 3.3b). This formalism is interesting because the topologies of biological brains, including the human brain, theoretically exist within its search space.

It is also possible to restrict substrate configurations to particular structural motifs to learn about their viability in isolation. For example, Churchland [Chu86] calls a single two-dimensional sheet of neurons that connects to another two-dimensional sheet a *state-space sandwich*. The sandwich is a restricted three-dimensional structure in which one layer can send connections only in one direction to one other layer. Thus, because of this restriction, it can be expressed by the single four-dimensional  $CPPN(x_1, y_1, x_2, y_2)$ , where  $(x_2, y_2)$  is interpreted as a location on the *target* sheet rather than as being on the same plane as the source coordinate  $(x_1, y_1)$ . In this way, CPPNs can search for useful patterns within state-space sandwich substrates (figure 3.3c).

Finally, the nodes need not be distributed in a grid. For example, nodes within a substrate that controls a radial entity such as a starfish might be best laid out with radial geometry, as shown in figure 3.3d, so that the connectivity pattern can be situated with perfect polar coordinates. The preliminary robot control experiment in Chapter 4 compares such a circular layout to a grid-based one.

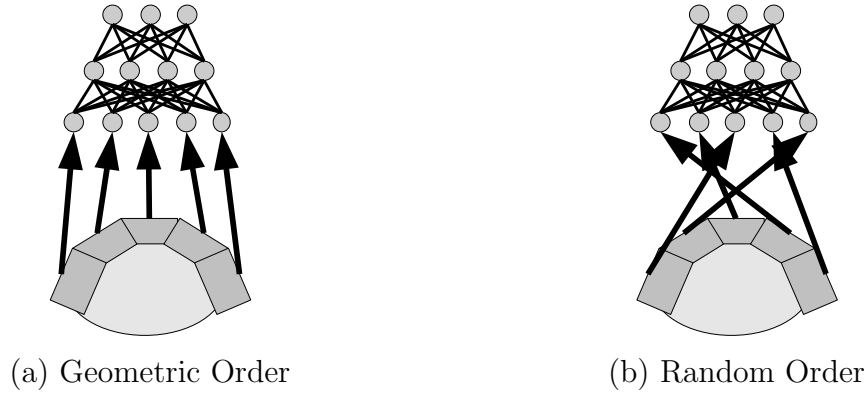


Figure 3.4: Regular ANN Input Order is Irrelevant. Even though the order of inputs to the ANN in (a) is correlated directly to the geometry of the front-facing robot sensors from which they are activated, traditional ANN learning algorithms such as backpropagation [RHW86] or traditional neuroevolution methods [SM02b, Yao99] are blind to such ordering. In fact, the arbitrary input order in (b) is identical from the perspective of the learning algorithm. Thus, counterintuitively, organizing inputs (or outputs) to respect task geometry provides no advantage with traditional ANN learning algorithms.

### 3.4 Input and Output Placement

Part of substrate configuration is determining which nodes are inputs and which are outputs. The flexibility to assign inputs and outputs to specific coordinates in the substrate creates an opportunity to exploit geometric relationships advantageously.

In many ANN applications, the inputs are drawn from a set of sensors that exist in a geometric arrangement in space. Unlike traditional ANN learning algorithms that are not aware of such geometry (as illustrated in figure 3.4), connective CPPN substrates *are* aware of their inputs' and outputs' geometry, and thus can use this information to their advantage.

By arranging inputs and outputs in a sensible configuration on the substrate, regularities in the geometry can be exploited by the encoding. There is room to be creative and try

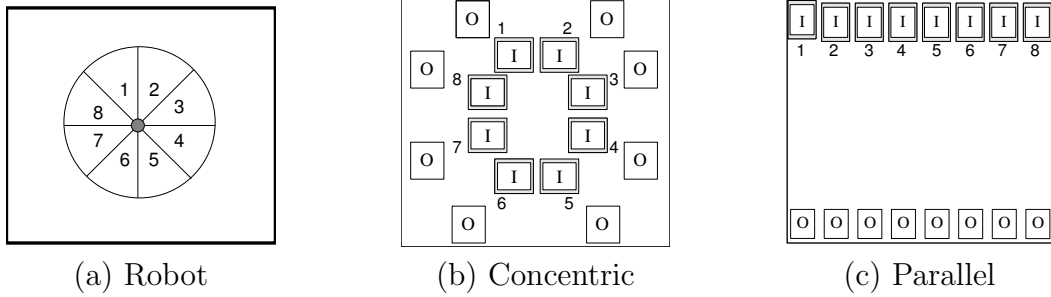


Figure 3.5: Placing Inputs and Outputs. A robot (a) is depicted with eight radar sensors along its circumference and eight motion effectors set at the same angle. In (b), the inputs (labeled *I*) and outputs (labeled *O*) are laid out literally according to the eight directions in space. In (c), the inputs are placed such that their location along  $x$  determines whether they represent a corresponding direction. Both arrangements create a geometric relationship between each input and its corresponding output. In this way, it is possible to give evolution a significant advantage from the start.

different configurations with different geometric advantages. For example, figure 3.5 depicts two methods by which the inputs and outputs of a circular robot can be configured, both of which create an opportunity to exploit a different kind of geometric relationship.

In one arrangement, the sensors on the circumference of the robot are arranged in a circle centered at the origin of the substrate, and outputs form a concentric circle around that (figure 3.5b). In this way, if the CPPN discovers radial symmetry or bilateral symmetry, it can use those coordinate frames to create a repeating pattern that captures regularities in the relationship between inputs and outputs. An alternate arrangement places the inputs and outputs on two parallel lines wherein equivalent horizontal position denotes equivalent angle (figure 3.5c). That way, evolution can exploit the similarity of horizontal positions. Both methods convey correspondence through a different geometric regularity.

By arranging neurons in a sensible configuration on the substrate, regularities in the geometry can be exploited by the encoding. Biological neural networks rely on such a

capability for many of their functions. For example, neurons in the visual cortex are arranged in the same retinotopic two-dimensional pattern as photoreceptors in the retina [CK04]. That way, they can exploit *locality* by connecting to adjacent neurons with simple, repeating motifs. Connective CPPNs have the same capability. In fact, geometric information in effect provides evolution with domain-specific bias, which is necessary if it is to gain an advantage over generic black-box optimization methods [WM97].

### 3.5 Substrate Resolution

As opposed to encoding a specific pattern of connections among a specific set of nodes, connective CPPNs in effect encode a general *connectivity concept*, i.e. the underlying mathematical relationships that produce a particular pattern. The consequence is that the *same connective CPPN* can represent an equivalent concept at different resolutions (i.e. different node densities). Figure 3.6 shows two connectivity concepts at different resolutions.

For neural substrates, the important implication is that the same ANN functionality can be generated at different resolutions. *Without further evolution*, previously-evolved connective CPPNs can be re-queried to specify the connectivity of the substrate at a new, higher resolution, thereby producing a working solution to the same problem at a higher resolution! There is no upper bound on substrate resolution, that is, a connectivity concept is infinite in resolution. While the higher-resolution connectivity pattern may contain artifacts

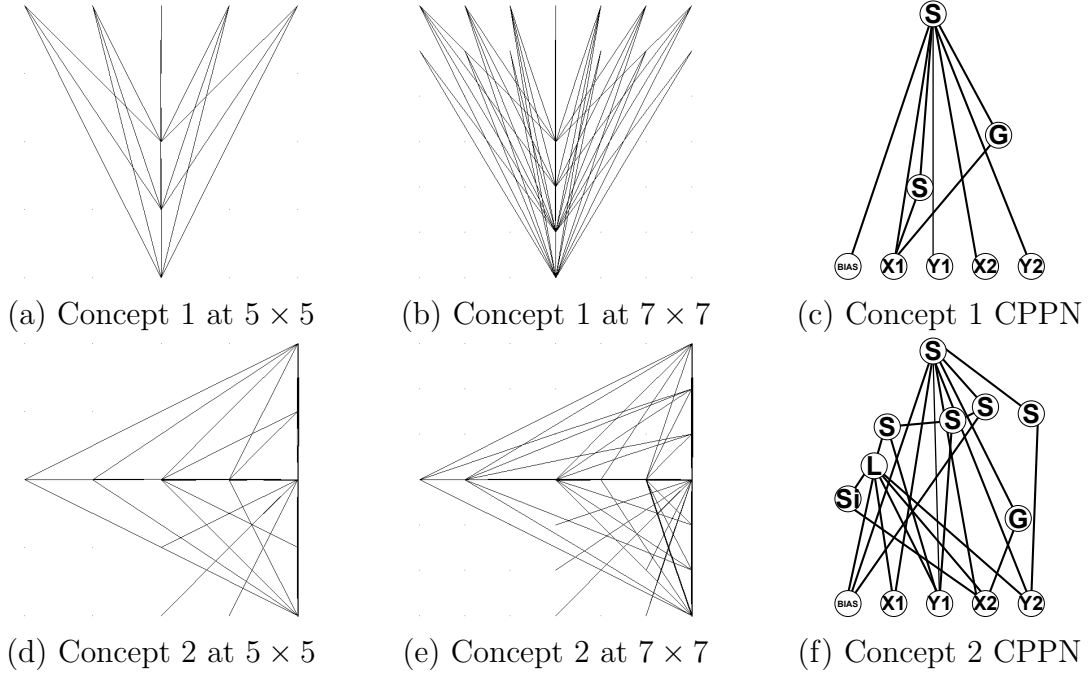


Figure 3.6: Equivalent Connectivity Concepts at Different Substrate Resolutions. Two connectivity concepts are depicted that were evolved through interactive evolution. The CPPN that generates the first concept at  $5 \times 5$  (a) and  $7 \times 7$  (b) is shown in (c). The CPPN in (f) similarly generates the second concept at *both resolutions* (d) and (e). This illustration demonstrates that CPPNs represent a mathematical concept rather than a single structure. Thus, the same CPPN can produce patterns with the same underlying concept at different substrate resolutions (i.e. different node densities). *CPPN activation functions in this dissertation are denoted by G for Gaussian, S for sigmoid, Si for sine, A for absolute value, and L for linear.*

that were not expressed at the lower resolution at which it was evolved, it will still embody a good approximation of the general solution at the higher resolution. Thus, increasing substrate resolution introduces a powerful new kind of complexification to ANN evolution. This capability will be exploited later in this dissertation to scale the size of multiagent teams.

### 3.6 Evolving Connective CPPNs

The approach in this dissertation is to evolve connective CPPNs with NEAT. This approach is called *HyperNEAT* because NEAT evolves CPPNs that represent spatial patterns in hyperspace. Each point in the pattern, bounded by a hypercube, is interpreted as a connection in a lower-dimensional connected graph. For example, a spatial pattern in a four-dimensional hypercube is interpreted as a two-dimensional connectivity pattern.

The basic outline of the HyperNEAT algorithm proceeds as shown in Algorithm 1.

In effect, as HyperNEAT adds new connections and nodes to the connective CPPN it is discovering new *global dimensions of variation* in connectivity patterns across the substrate. Early on it may discover overall symmetry, whereas later it may discover the concept of receptive fields. Each new connection or node in the CPPN represents a new way that an entire pattern can vary, i.e. a new regularity. Thus HyperNEAT is a powerful approach to evolving large-scale connectivity patterns and ANNs.



---

**Algorithm 1** HyperNEAT

---

1. Choose substrate configuration (i.e. node layout and input/output assignments)
  2. Initialize population of minimal CPPNs with random weights.
  3. Repeat until solution is found:
    - (a) For each member of the population:
      - i. Query its CPPN for the weight of each possible connection in the substrate. If the absolute value of the output exceeds a threshold magnitude, create the connection with a weight scaled proportionally to the output value (figure 3.1).
      - ii. Run the substrate as an ANN in the task domain to ascertain fitness.
    - (b) Reproduce the CPPNs according to the NEAT method to produce the next generation population.
- 

The next chapter details an initial demonstration of HyperNEAT that evolves the controller for a single agent performing a food-gathering task, providing not only a test of the HyperNEAT method, but a first step towards evolving controllers for multiple agents.

## CHAPTER 4

### SINGLE-AGENT HYPERNEAT

If sensors and effectors are placed such that they respect regularities in the outside world, HyperNEAT can discover those regularities through connective CPPNs and exploit them to solve the problem. For example, HyperNEAT can discover that the way one reacts to stimulus on the left is related to the way one react to similar stimulus on the right. In this way, HyperNEAT can solve problems with high-dimensional input because it does not need to learn the meaning of each sensor independently.

The following set of experiments demonstrate this capability and its implications through a food gathering task. This task was chosen for its simplicity as a proof-of-concept; it effectively isolates the issue of sensor and output placement, which is a primary advantage of HyperNEAT. Furthermore, although the task is simple, at high resolutions (i.e. with large numbers of sensors and effectors) it becomes a difficult optimization problem because of its increasing dimensionality. In the experiments, two different sensor placement arrangements are compared that both present a chance to exploit regularity in different ways.

## 4.1 Food Gathering Experiment

The food gathering domain works as follows. A single piece of food is placed within a square room with a robot at the center (as in figure 3.5a). A set of  $n$  rangefinder sensors, placed at regular angular intervals, encircle the robot’s perimeter. The robot has a compass that allows it to maintain the same orientation at all times, that is, its north-facing side *always* faces north and never rotates. Internally, the robot also contains a set of  $n$  effectors. Each effector, when activated, causes the robot to move in one of  $n$  directions. Thus, there is one effector for each sensor that points in the same direction. The robot’s objective is to go to the food.

The interpretation of effector outputs constrains the problem and the potential solutions. For the experiments in this section, the motion vector resulting from effector output is interpreted to incentivize HyperNEAT to find holistic solutions, i.e. solutions that do not only require a single connection. The robot moves in the direction corresponding to the largest effector output. In the case of a tie, the robot moves in the direction of the first tied output starting from  $45^\circ$  above west and moving clockwise. The robot’s speed  $s$  is determined by  $s = (s_{max} o_{max}) (\frac{o_{max}}{o_{tot}})$ , where  $s_{max}$  is the maximum possible speed,  $o_{max}$  is the maximum output, and  $o_{tot}$  is the sum of all outputs. The first term correlates speed with output, such that to go the maximum speed, the robot must maximize the output corresponding to the direction of the food. The second term encourages the robot to excite a single output by penalizing it for activating more than one at a time. Furthermore, outputs

have *sigmoidal activation*, which means that if their input is zero, they will output 0.5. Thus the robot also needs to inhibit effectors that point in the wrong direction because they will otherwise slow down motion in the chosen direction. This convention also discourages relying on tie-breaking by making it less efficient. Thus, while diverse solutions still work in this domain, many are not optimal in terms of speed. The best solutions require a correct pattern connecting to all the outputs from all the sensors.

Each robot attempts  $r$  trials, where  $r$  is twice the resolution; thus higher resolutions are evaluated on more trials. For each trial a single piece of food is placed 100 units away from the robot at either the center of a sensor or the border between two sensors. Each trial tests a different such location. If a robot is not able to get food for a particular trial after 1,000 ticks, its trial ends. Individuals are evaluated based on their amount of food collected and the average speed at which they obtain each item:  $fitness = (10,000 \frac{f_c}{r}) + (1,000r - t_{tot})$ , where  $f_c$  is the total number of food collected and  $t_{tot}$  is the total time spent on all trials.

This task is a good proof of concept because it requires discovering the underlying regularity of the domain: Two nodes at the same angle (i.e. the sensor and effector) should be connected and the others inhibited. If this concept is discovered, the task is effectively trivial. However, direct encodings would need to discover the connection between each pair *independently*. That is, they cannot discover the underlying *concept* that describes the solution. In contrast, HyperNEAT can discover the general concept. Demonstrating this fact helps to establish the promise of HyperNEAT in more complex domains.

### 4.1.1 *Sensor Placement*

As discussed in Section 3.4, HyperNEAT makes it possible to decide how the sensors and effectors should be placed in the substrate to allow the concept to be discovered. In general, the geometry of the placement scheme should reflect the correlation between sensors and effectors. Two different sensor placements and substrate configurations are attempted:

1. **Two concentric circles of nodes (figure 3.5b).** The inner circle (radius .5) is the sensors and the outer circle (radius 1) is the effectors. The nodes are placed at the same angle as they exist in the robot. In this layout, the key regularity is captured by shared angle. It is also interesting because the sensors and effectors are placed exactly in the shape of the robot, an intuitive scheme that would not be meaningful to traditional methods.

2. **Two parallel lines of nodes (figure 3.5c).** The top row of sensors are placed in clockwise order starting from the sensor closest to  $45^\circ$  above west. In the bottom row, effectors are placed in the same order. In this way, the key regularity is geometrically expressed as two nodes being in the same column.

### 4.1.2 *Additional Geometric Bias*

Connective CPPNs already provide a strong geometric bias, but they also make possible injecting additional *a priori* knowledge about the problem into the search. In biology, distance

is a significant bias simply because in the physical world it is easier to connect to something closer than farther away. Thus, if a CPPN is *given* connection length as an input, related concepts placed close together on the substrate can be treated specially by the CPPN.

In *both* placement schemes, sensors should excite their nearest neighbor effector and inhibit those farther away. While CPPNs have the capability to discover the concept of distance themselves, they may solve tasks more efficiently if it is given. To explore this capability, a separate experiment is performed on both substrate configurations in which the CPPN receives an extra input representing the Euclidean distance between the two points being queried.

#### 4.1.3 *Scaling*

An important question is whether HyperNEAT will discover the key regularity and whether one configuration has an advantage over another. Furthermore, *if* HyperNEAT does discover the underlying concept, then solutions should scale to more sensors and effectors without further evolution simply by increasing the resolution of the substrate.

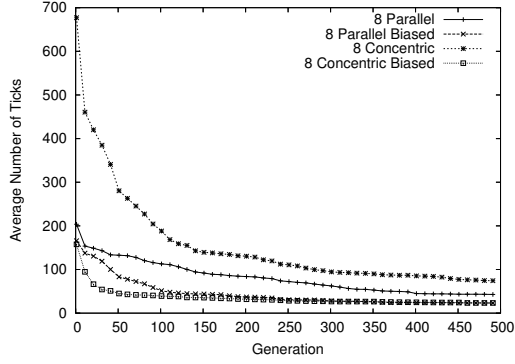
Therefore, the number of inputs and outputs of the champion of each generation from all runs of both configurations is doubled several times starting from the  $8 \times 8$  resolution on which it is evolved. Each double-resolution substrate samples twice as many points as its predecessor by decreasing the angular sampling interval around the robot by half. Doubling

for each champion proceeds from the initial  $8 \times 8$  resolution to a maximum size of  $128 \times 128$ , a 16-fold increase in resolution.

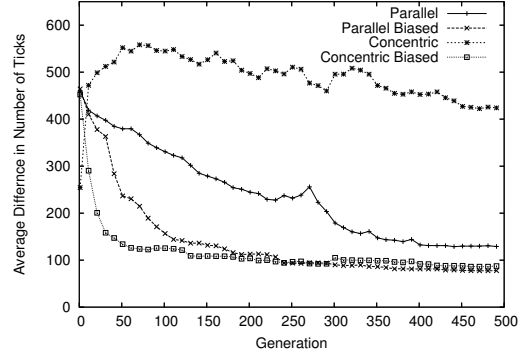
## 4.2 Food Gathering Results

All sensor configurations were able to collect food at all positions within the first few generations except unbiased concentric, which took on average 33 generations to learn how to get food at every position. Thus, for most configurations, the main challenge was to learn to get food *efficiently*. The performance measure in this section is thus the average time (i.e. number of ticks) it takes the robot to get a piece of food over all its trials. Robots that cannot get the food in a trial are given the maximum time 1,000 for that trial. Results are averaged over 20 runs.

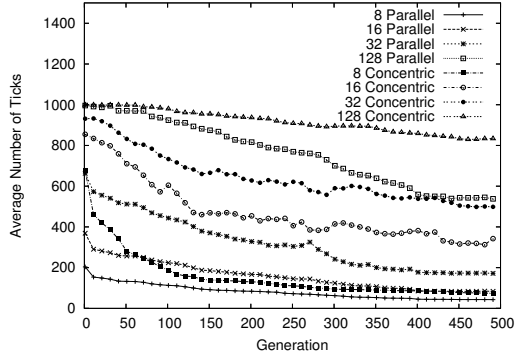
Figure 4.1a shows how performance improved over 500 generations for both placement schemes, with and without the length-input geometric bias. Parallel placement on average evolved significantly faster strategies ( $p < 0.05$ ) than concentric. However, the geometric bias significantly increased performance of both methods after the fourth generation ( $p < 0.01$ ). Furthermore, the geometric bias is so useful that it erases the difference between the two schemes, causing both to perform similarly when present. Thus, parallel placement is easier to exploit for HyperNEAT except when the CPPN is provided connection length as input.



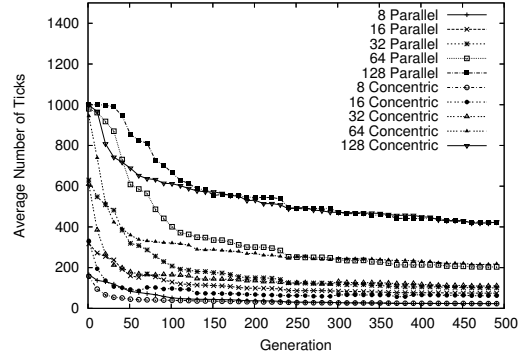
(a) Initial Performance



(b) Scalability



(c) Unbiased Scaling



(d) Biased Scaling

Figure 4.1: HyperNEAT Performance. The performance of both sensor layouts at resolution eight with and without biasing is shown in (a). The difference in speeds when different methods are scaled from 8 to 32 is shown in (b). Graphs (c) and (d) show the speeds of the two sensor placement schemes at all resolutions with and without biasing, respectively. The conclusion is that HyperNEAT learns to exploit the placement geometry.



### 4.2.1 *Scaling Performance*

As described in Section 4.1.3, after evolution at resolution eight is completed, generation champions are reevaluated by doubling their resolution repeatedly without further evolution. Individuals generally *did* retain the ability to collect food although there is some degradation in performance at each increment in resolution. To illustrate this degradation for different configurations, figure 4.1b shows the average *difference* in efficiency between resolution eight and resolution 32; lower numbers imply better ability to scale.

Parallel placement scaled significantly more effectively than concentric except in the earliest generations ( $p < 0.01$ ; Student's t-test). However, as with efficiency, when concentric placement's CPPN was provided length as input, it scaled as well as parallel placement did with length input. In both biased cases, scaling was significantly better than runs using concentric placement without the geometric bias ( $p < 0.01$ ), but not significantly over unbiased parallel placement.

As figure 4.1b shows, unbiased concentric placement degraded significantly between resolution eight and 32; in fact individuals could no longer collect food in every position. However, it turns out that information about the task was still retained implicitly at the higher resolution: When allowed to *continue* evolving at the higher resolution, solutions that collect all the food were always found within five generations (2.5 on average). On the other hand, when concentric evolution is started from scratch at a *lower* resolution, it takes on average 33 generations to learn to get food on every trial. Thus, even when performance

degrades significantly after scaling, the resultant individual still retains important geometric information that can be quickly tweaked to work at the higher resolution.

Figure 4.1c shows the average absolute performance at different resolutions for CPPNs without the geometric bias, and figure 4.1d shows the same comparison for those with the bias. Parallel placement consistently outperformed concentric on the same resolution (figure 4.1c). However, again, when provided the length input (figure 4.1d), the performance of the two placement schemes no longer significantly differed. Although each increment in resolution leads to a graceful degradation in performance, scaled individuals at all resolutions and in all configurations significantly outperformed a set of random individuals, showing further that scaling indeed retains abilities present in the lower-resolution network ( $p < 0.01$  for all scaled performance versus random). Furthermore, remarkably, although *average time* degrades, most scaled networks could still collect all the food if their unscaled predecessor could.

In a more dramatic demonstration of scaling, one high-fitness individual of each sensor placement scheme was scaled to a resolution of 1,024. The resulting ANNs each had over *one million* connections and could still gather all the food.

### 4.2.2 *Repeating Motifs*

To consistently activate highly the one output pointing in the right direction and inhibit all others requires discovering such a *motif* and repeating it across the substrate. Figure 4.2 shows several examples of this motif at different locations in both concentric and parallel substrates. The benefit of CPPN representation is that it need only discover the correct motif *once* by exploiting how it relates to the geometry of inputs and outputs.

The CPPN in figure 4.2a (top row) shows how such exploitation is possible. HyperNEAT discovered that by connecting  $x_1$  to a Gaussian node with a positive weight and  $x_2$  to the same node with a negative weight, the weight connecting any two nodes in the substrate is made a function of horizontal distance. This principle is all that is necessary to cause the same motif to appear at every locus of horizontal correlation.

On the other hand, the principle of radial symmetry that underlies the correct motif in concentric substrates is more challenging. Angular differences necessary to exploit concentric regularity take more structure to compute, although the concept is eventually discovered.

### 4.2.3 *Scaling Analysis*

Figure 4.3(a and b) shows evolved parallel and concentric motifs at both  $8 \times 8$  and  $16 \times 16$  resolution.

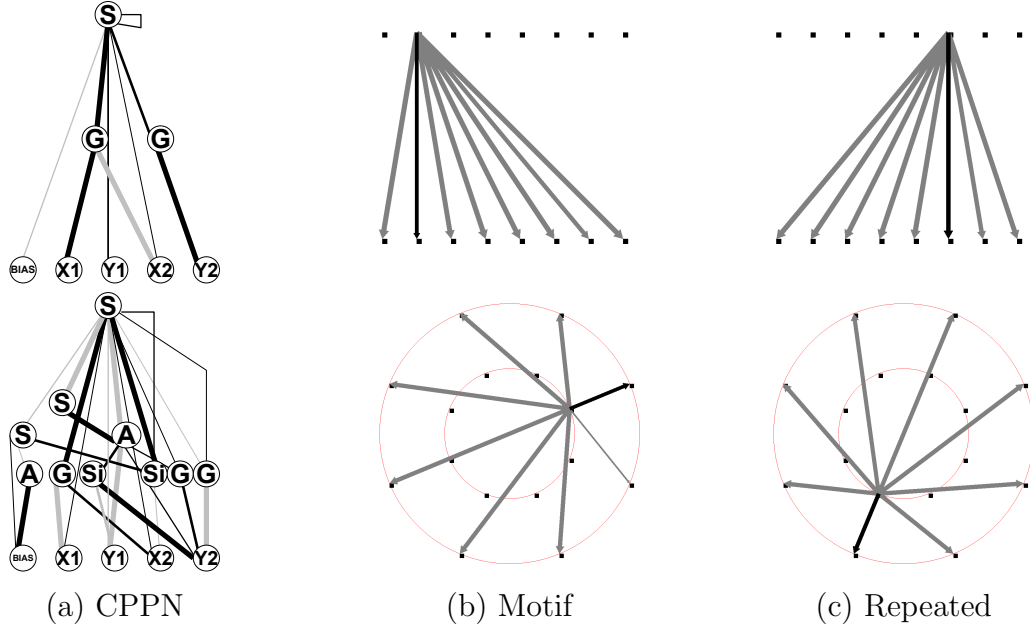


Figure 4.2: Repeated patterns in solutions. The CPPNs in (a) represent parallel (top row) and concentric (bottom row) substrates that solve the task. The connectivity patterns in (b) and (c) are outgoing *motifs*, each from a single input node in the substrate. These images show that the same motif is repeated in different locations.

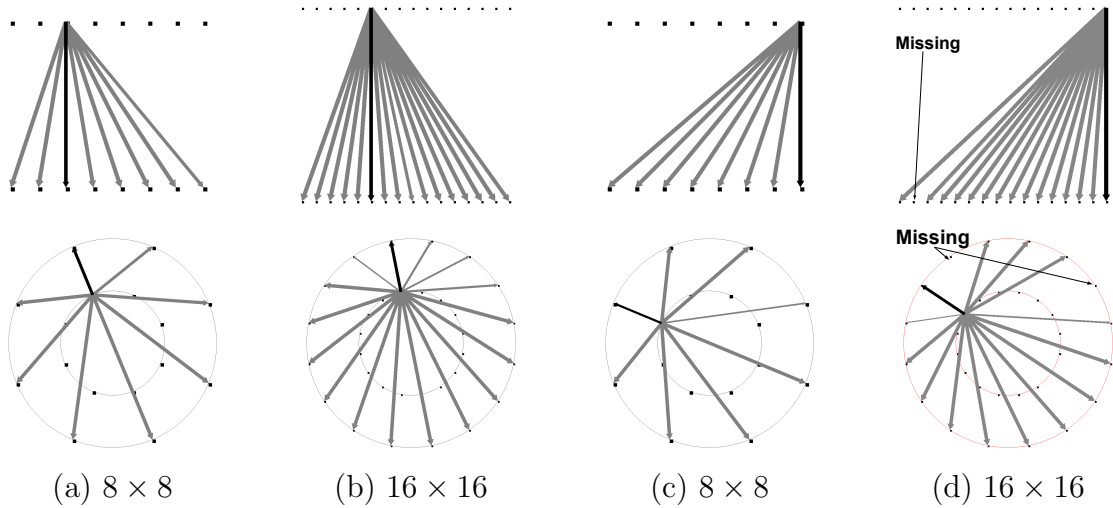


Figure 4.3: Perfect and Imperfect Scaling. The parallel (top row) and concentric (bottom row)  $8 \times 8$  motifs in (a) scale perfectly to the  $16 \times 16$  motifs in (b). However, because of artifacts at higher resolution, not all motifs scale perfectly, as shown between (c) and (d), in which missing connections are identified by arrows. While the motif is mostly intact, slight imperfections of this type are common during scaling in this task.

Scaling up can sometimes produce small artifacts that are not apparent at lower resolutions (figure 4.3c and d). These imperfections vary greatly as do their effect on performance, although in all cases sufficient information was retained to quickly recover the old behavior in under five generations.

Analyzing the source of higher-resolution imperfections explains how CPPNs represent regularities. Recall that a connective CPPN is in effect a pattern within a four-dimensional hypercube. If two of those dimensions, e.g.  $x_1$  and  $y_1$ , are fixed, then the remaining two dimensions define a two-dimensional cross-section of the hypercube. The pattern within that cross-section is in effect the infinite-resolution connectivity pattern for one input node from which all finite substrate resolutions are sampled. Figure 4.4 depicts two such cross-sections for both concentric and parallel substrates. The figure shows that the CPPN constructs a *pattern* that, when sampled at the right resolution, defines the connectivity weights of all connections between the fixed source location and the sample point.

Because only the sample points are required to be correct during evaluation, the unsampled portion of the pattern can exhibit artifacts that are not visible at the sample resolution, but may become so at higher resolutions. However, interestingly, the pattern often elegantly captures the fundamental geometric relationship, as in figure 4.4a and figure 4.4c, which explains why scaling sometimes is perfect or close to perfect.

Thus, one conclusion on scaling is that depending on the task, it always retains *at least* what was known at the lower resolution, which is significantly better than starting from scratch if evolution is to continue at the higher resolution.

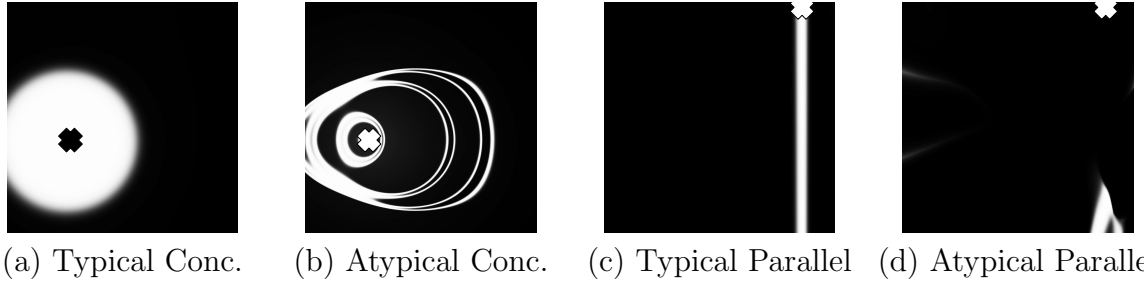


Figure 4.4: Hypercube Cross-Sections. Typical (a and c) and atypical (b and d) two-dimensional cross-sections of hypercubes generated by CPPNs are depicted for working concentric (a and b) and parallel (c and d) substrate configurations. Each cross-section represents the infinite-resolution outgoing connectivity pattern originating from the location of the “X.” When substrates are generated in practice, these cross-sections are sampled at the substrate resolution. Thus, much of the detail in the patterns is discarded for low-resolution substrates, though it may reemerge when scaling to higher resolution.

#### *4.2.4 Evolving Motifs*

Rather than finding the value of each connection separately, evolution in HyperNEAT progresses by discovering global regularities. Figure 4.5 traces progress over parallel and concentric runs of evolution. The early solutions are simple linear combinations of the coordinates. Although inefficient, these patterns learn to gather food through a few good connections that compensate for others. Later in evolution, symmetries and regularities begin to be discovered, although they are not always the most fundamental to the task. For example, the concentric substrate in figure 4.5b partially solves the task by exploiting its bilateral symmetry, even though the task is more fundamentally radially symmetric. Finally, evolution discovers the essential regularity of the task domain, which is horizontal distance for parallel substrates and angular disparity in concentric substrates (figure 4.5c).

The next section discusses the major ramifications of this work.

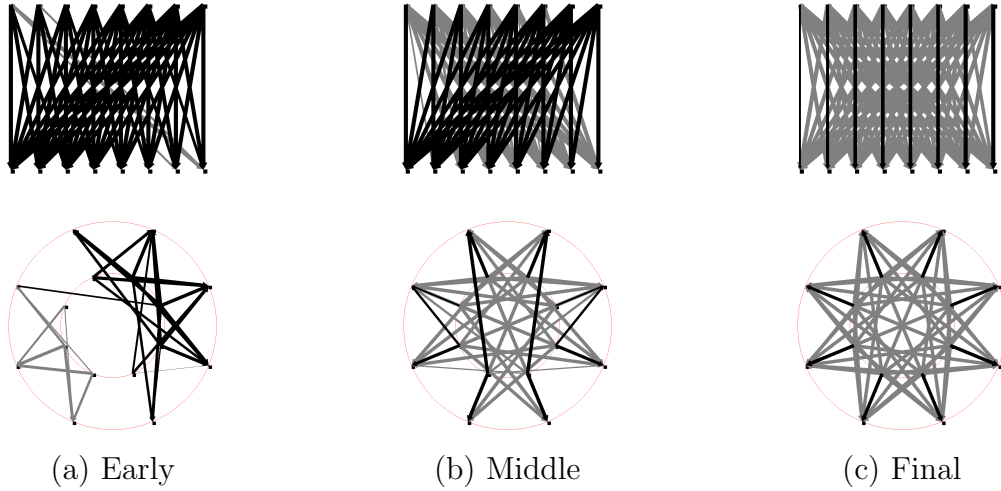


Figure 4.5: Evolution Discovers Regularities. Early-generation connective CPPNs typically produce simple connectivity patterns (a). Eventually, HyperNEAT begins to exploit regularities (b), though they may not be the most fundamental ones. Finally, HyperNEAT discovers the fundamental regularity that underlies the task for the given substrate configuration (c). Thus, instead of optimizing individual connection weights, evolution is a process of gradually discovering and exploiting holistic regularities.

### 4.3 Single-Agent Experiment Discussion

The central insight in HyperNEAT is that regularity in higher-dimensional spatial patterns is easily mapped to regular connectivity in lower-dimensional space, creating a new kind of indirect encoding for ANNs. Experimental results show how it is possible to encode over a million connections by discovering the pattern of recurring motifs underlying a solution. This section explores the implications of this capability and its underlying methodology.

### 4.3.1 Substrate Scaling

While not always perfect, the ability to scale without further evolution is a significant advance for neuroevolution. The important consequence is that information learned at the lower resolution is retained at the higher resolution; it is only within the expanded portions that error may be introduced. Thus, the real benefit of this capability is that in the future, further evolution can be performed at the higher resolution. Instead of starting over from scratch, HyperNEAT can continue to build on lower-resolution solutions after they are magnified. The number of sensors, hidden nodes, and outputs can be multiplied without losing prior knowledge because CPPNs in effect *decouple* solutions from individual inputs and outputs.

### 4.3.2 Substrate Configuration and Geometry

The food gathering task demonstrates that, in contrast to regular ANNs, different substrate placement schemes create geometric relationships that are exploitable in different ways, some more easily than others. Connective CPPNs exploit regularities in parallel placement more efficiently than in concentric because the angular differences necessary to exploit concentric regularity take more structure to compute with the given set of activation functions. Thus the activation functions in the CPPN are like a *language* that describes geometric structure. With the right words, it is easier to describe a particular relationship. HyperNEAT allows



the experimenter to inject knowledge into the search through such configurations, e.g. by grouping correlated objects or arranging sensors analogously to their real world geometry.

Furthermore, additional geometric information provided as input to the connective CPPN can significantly simplify the structure necessary to describe key motifs. When *connection length* is provided as input, both placement schemes produce equivalent results. This result is important because it shows that connective CPPNs allow the experimenter to provide *hints* to the learning algorithm about the kinds of geometric principles that may underly the solution.

It is now possible to exploit the geometry of many domains and tasks for the first time. Thus, by applying HyperNEAT to control tasks, new kinds of solutions may be discovered that genuinely exploit regularities in such tasks. This idea is extended in this dissertation to exploit geometry in multiagent learning tasks, which is explained in the next chapter.

## CHAPTER 5

### APPROACH: MULTIAGENT HYPERNEAT

As shown in the previous chapter, HyperNEAT can effectively encode the behavior of a single agent. The focus of this dissertation, however, is to extend HyperNEAT so that it can encode the behaviors of an entire team of agents. This method, called multiagent HyperNEAT, provides a new perspective on the problem of multiagent learning by focusing on the relationships among agent policies rather than on the policies of individuals. Recall that the *policy geometry* of a team is the relationship between the canonical starting positions of agents on the field and their behavioral policies. Multiagent HyperNEAT is based on the idea that policy geometry is the right level of description for a team because it can be encoded naturally as a pattern.

Multiagent HyperNEAT can find and exploit the regularities in a team’s policy geometry for fast, efficient multiagent learning. Recall the soccer team from figure 1.1. The policies of the players become progressively more defensive the closer they are to the goal. This geometric relationship can be represented as a pattern that starts at center field, and increases as it moves towards the goal. Furthermore, HyperNEAT can encode patterns that repeat, yet also vary (e.g. figure 3.2). In multiagent learning this capability is beneficial for discovering teams at the right point on the *continuum of heterogeneity*, wherein the degree of variation

represents the heterogeneity of the team. Traditional multiagent learning approaches have no facility for capturing such regularities, highlighting the unique contribution of HyperNEAT.

To understand how the policy geometry of a team can be encoded, it helps to begin by considering *homogeneous teams*, which in effect express a trivial policy geometry in which the same policy is uniformly distributed throughout the team at all positions. Thus this section begins by exploring how teams of purely homogeneous agents can be evolved with an indirect encoding, and then transitions to evolving heterogeneous teams that are represented by a single genome in HyperNEAT. Later chapters then enhance the formalism developed in this chapter to allow scaling.

## 5.1 Pure Homogeneous Teams

A homogeneous team only requires a single controller that is copied once for each agent on the team. To generate such a controller, a four-dimensional CPPN with inputs  $x_1, y_1, x_2$ , and  $y_2$  (figure 5.1a) queries the substrate shown in figure 5.1c, which has five inputs, five hidden nodes, and three output nodes, to determine its connection weights. This substrate is designed to geometrically correlate sensors to corresponding outputs (e.g. seeing something on the left and turning left), which allows the CPPN to exploit the geometry of the agent as shown in Chapter 4. However, the agents themselves have exactly the same policy no matter

where they are positioned. Thus while each agent is informed by geometry, their policies cannot differentiate genetically.

## 5.2 Teams on the Continuum of Heterogeneity

Heterogeneous teams are a greater challenge; how can a single CPPN encode a *set* of networks in a pattern, all with related yet varying roles? Generative systems such as HyperNEAT are naturally suited to capturing such patterns by encoding the policy geometry of the team as a pattern. The remainder of this section discusses a method by which HyperNEAT can encode such teams.

The main idea is to place the whole *set* of networks on the substrate and compute their connection weights as *both* a function of their location within each network *and* within the larger pattern of multiple networks. The CPPN then queries all the connection weights in this multiagent substrate (figure 5.1d), which contains five copies of the homogeneous substrate, one for each agent. Notice that in effect this arrangement assigns a position to each network within this policy geometry.

The challenge for such a substrate is to tell the CPPN where one agent stops and another begins; the CPPN could learn this pattern, but it is more effective to tell it where each agent is from the start. For this purpose, two special function nodes are added to the initial CPPN (whose topology later evolves through HyperNEAT; figure 5.1b) so that each receives input

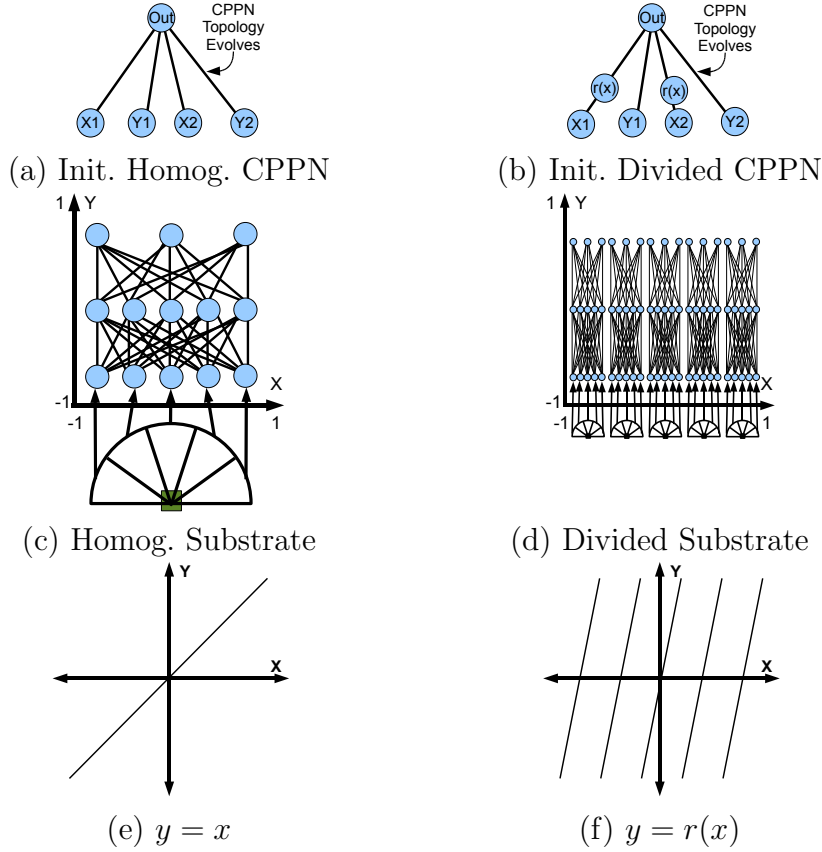


Figure 5.1: Multiagent HyperNEAT. This figure depicts the CPPNs and substrates that encode multiple agents with HyperNEAT. The CPPN in (a) generates a single controller for a single agent or a homogeneous team of agents. The single controller substrate that is queried by this CPPN is shown in (c). In contrast, the CPPN in (b) encodes a heterogeneous team by sampling the substrate in (d), which is made up of the single substrate (c) copied five times, but repeated across the substrate. The  $r(x)$  nodes above  $x_1$  and  $x_2$  in the initial heterogeneous CPPN (b) repeat the  $x$  coordinate frame (e), duplicating it for each agent (f) while also maintaining a global coordinate system through  $x_1$  and  $x_2$ . In this way, the CPPN can create patterns across both the agents' bodies and the team as a whole. If the same point is sampled within any two agents in (e),  $r(x)$  will return the same value (though  $x$  will not), giving agents on the team their own coordinate frame. Note that CPPNs depicted in (a) and (b) increase in complexity over evolution through the NEAT algorithm.

from either  $x_1$  or  $x_2$  and outputs a coordinate frame,  $r(x)$ , that repeats the same set of coordinates once per agent (figure 5.1f), thereby telling the CPPN where each node is *within* each individual agent. That is,  $r(x)$  compresses  $y = x$  and repeats it the same number of times as there are agents on the team. The CPPN thus sees *both* the coordinate frame  $r(x)$  within each agent and the coordinate frame  $x$  of the entire team (from CPPN inputs  $x_1$  and  $x_2$ ). In this way, it can simultaneously encode shared patterns within agents (i.e. by basing them on the repeating coordinate frame) and patterns that vary across teams (e.g. agents on the left can mirror the behaviors of agents on the the right through a symmetric function of absolute position).

The heterogeneous substrate formalizes the idea of encoding a team as a pattern of policies. This capability is powerful because generating each agent with the same CPPN means they can share tactics and policies while still exhibiting variation across the policy geometry. In other words, policies are spread across the substrate in a pattern just as role assignment in a human team forms a pattern across a field. However, even as roles vary, many skills are shared, an idea elegantly captured by indirect encodings. The complete multiagent HyperNEAT algorithm is enumerated in Algorithm 2.

---

**Algorithm 2** Multiagent HyperNEAT

---

1. Set the substrate to contain the necessary number of agents (figure 5.1d).
  2. Initialize a population of minimal CPPNs (figure 5.1b) with random weights.
  3. Repeat until a solution is found or the maximum number of generations are reached:
    - (a) For each CPPN in the population:
      - i. Query it for the weight of each connection in the substrate within each agent’s ANN. If the absolute value of the output exceeds a threshold magnitude, create the connection with a weight scaled proportionally to the output value (figure 3.1).
      - ii. Assign the generated ANNs to the appropriate agents and run the team in the task domain to ascertain fitness.
    - (b) Reproduce the CPPNs according to the NEAT method to create the next generation’s population
- 

### 5.3 Seeding

Seeding each agent of a multiagent team with the behavior of a single pre-trained agent exploits the fact that effective teams often lie close to pure homogeneity on the continuum of heterogeneity. That is, most teams include agents that share a number of skills. For example, all members of a soccer team know how to kick, pass, and dribble a ball. Thus it would be inefficient if each agent had to learn these basic skills separately. In this way, training a single agent to master the core skill set is not only computationally more efficient (only one agent needs to be simulated), but generally leads to more efficient learning of team-wide strategies. While most multiagent learning methods can seed teams, because they are agnostic to the team policy geometry, they cannot subsequently vary the seed policy along the policy geometry.

In contrast, multiagent HyperNEAT creates teams as a function of their policy geometry and can alter the seed policy by gradually shifting it away from pure homogeneity along the continuum of heterogeneity. The challenge for multiagent HyperNEAT is to make a strong single policy represent the policies of an entire team. Pure homogeneous teams are trivial to seed because such teams only require one controller that is copied to each member of the team; thus seeding the team requires only that evolution is started with a population of controllers derived from the seed. However, the same method does not work with heterogeneous teams because a CPPN that represents a single agent does not automatically represent a whole team. Thus it is necessary to slightly alter the CPPN that represents a single agent’s controller so that it can encode the controllers of every agent on a team.

Recall that the CPPN that represents a single agent takes the four inputs  $x_1$ ,  $y_1$ ,  $x_2$ , and  $y_2$  (figure 5.1a). The substrate also requires the  $r(x)$  function to divide the agents. Thus  $r(x)$  must be *added* to the seed CPPN such that the seed policy is preserved. The  $r(x)$  function is intended to repeat once per agent. If there is only one agent,  $r(x)$  need only repeat once, in which case  $r(x) = x$ . Thus a single-controller substrate is just a special case of the heterogeneous substrate. Therefore, by diverting the  $x_1$  and  $x_2$  inputs in the single agent CPPN (figure 5.2, left) to new  $r(x)$  nodes, the CPPN generates a repeating connectivity pattern in the substrate. All connections originally leaving  $x_1$  and  $x_2$  are then moved to project from their respective  $r(x)$  nodes (figure 5.2, right). Each agent in the team thereby becomes a copy of the seed policy because nothing initially connects  $x_1$  and



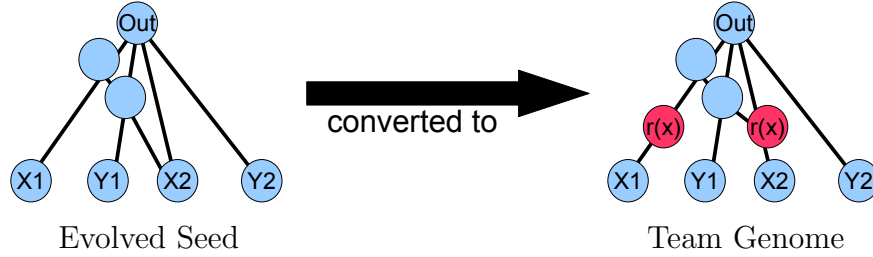


Figure 5.2: Heterogeneous seeding. From a CPPN that generates a successful single agent, multiagent HyperNEAT can generate a team of agents. To allow the CPPN to differentiate between team members, the CPPN at left is modified. The CPPN is altered so that the  $x_1$  and  $x_2$  inputs feed into two new nodes (shown at right). All connections that originally projected from the inputs are changed to project from these new nodes (darkened) instead. This change maintains the coordinate frame of a single agent in the  $r(x)$  nodes, while the  $x_1$  and  $x_2$  inputs now denote the coordinate frame of the *team*, allowing the CPPN to generate patterns relevant to both.

$x_2$  to the rest of the CPPN (which would allow variation over the policy geometry) except through  $r(x)$ . During evolution, however, mutations can connect the absolute  $x$  coordinates to the rest of the network, yielding increasingly heterogeneous behavior. This technique works because the coordinate system defined by  $r(x)$  for each agent is *exactly the same* as the single agent’s coordinate system (figure 5.1e), except that now it repeats several times across  $x$  (figure 5.1f).

The next chapter describes a multiagent experiment designed to establish the promise of the overall multiagent HyperNEAT approach introduced so far.

## CHAPTER 6

### MULTIAGENT PREDATOR-PREY EXPERIMENT

This chapter describes a multiagent HyperNEAT experiment designed to validate the approach in a predator-prey domain. Details of the experiment were first published by myself and Kenneth Stanley at the Genetic and Evolutionary Computation Conference (GECCO-2008) [DS08] and won the best paper award in the Generative and Developmental Systems track.

#### 6.1 Predator-Prey Experiment

The aim of the experiment is to establish the potential of representing a team as a pattern of policies. Cooperative multiagent predator-prey is a good platform to test this idea because the task is challenging yet easy to understand. Through an indirect encoding like multiagent HyperNEAT, the hope is that tightly coordinated agent policies can be encoded as a pattern.

In the version of predator-prey in this section, agents cannot see their teammates, and because prey run away from nearby predators, it is easy for one predator to undermine another's pursuit by knocking its prey off its path. Therefore, predators must learn consistent

roles that complement those of their allies. At the same time, agents need basic skills in interpreting and reacting to their sensors. Because multiagent HyperNEAT creates all the agent ANNs from the *same* CPPN, it has the potential to balance these delicate ingredients.

For the experimental parameters see the Appendix.

### 6.1.1 *Predators and Prey*

Each predator agent on the team is controlled by the ANN in figure 6.1b. Predators are equipped with five rangefinder sensors spanning a  $180^\circ$  arc that detect prey within 300 units. Their goal is to capture (i.e. intercept) the prey agents by positioning themselves so that a prey is visible to their front sensor and less than 25 units away (this area is shown as a shaded cone in figure 6.1a). Predators *cannot* sense each other.

At each discrete moment of time, a predator can turn up to  $36^\circ$  and move up to five units forward. The number of units moved is  $5F$ , where  $F$  is the forward effector output. The predator also turns by  $(L - R) \times 36^\circ$ , where  $L$  is the left effector output and  $R$  is the right effector output. A negative value is interpreted as a right turn.

Prey agents are programmed to maintain their current location until they are threatened; if there is a predator within 50 units the prey moves in the opposite direction of the closest predator. Prey move at the maximum speed of predator agents. That way, it is impossible for a single predator to catch a prey.

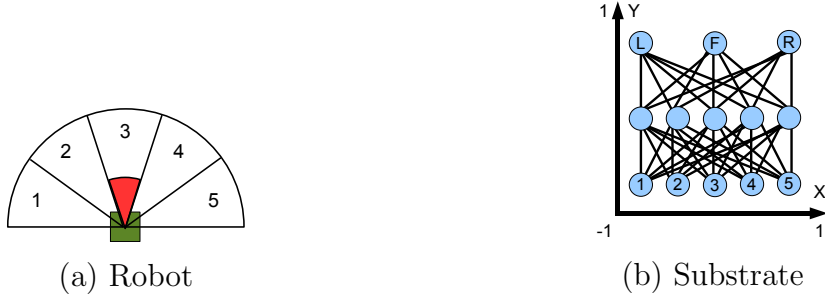


Figure 6.1: Single Predator Substrate Configuration. An autonomous robot (a) is equipped with five sensors, spanning a  $180^\circ$  arc in front of it and labeled 1 through 5 from left to right. The substrate that controls the robot (b) is arranged such that the placement of inputs in the ANN corresponds to the physical locations of the sensors on the robot (e.g. the leftmost sensor corresponds to the leftmost input). Similarly, the outputs of the network are related to their effects on the agent and correspond to the sensors (e.g. the left turn output is on the left side of the network and above the leftmost sensor input). Such placement allows the CPPN to easily generate connectivity patterns that respect the geometry of the problem, such as left-right symmetry.

The predator team starts each trial in a line, spaced 100 units apart, facing the prey (figure 6.2). The environment the agents inhabit is physically unbounded, and each trial lasts 1,000 time steps. At the end of each trial the team receives a score of  $10,000P + (1,000 - t)$ , where  $P$  is the number of prey captured and  $t$  is the time it takes to catch all the prey. If all prey are not captured,  $t$  is set to 1,000. Team fitness is the sum of the scores from two trials on which the team is evaluated. This fitness function encourages the predators to capture all the prey as quickly as possible.

The major challenge for the predators is to coordinate despite their inability to see one another. This restriction encourages establishing *a priori* policies for cooperation because agents thus have little information to infer each others' current states. Such situations are not

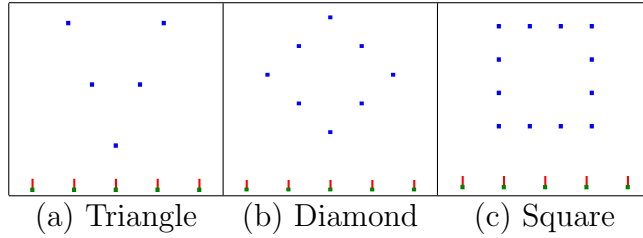


Figure 6.2: Prey Formations. The prey (upper agents) can be arranged in three formations. The predators (lower agents) are always placed in the same evenly-spaced line below the prey. Each formation presents a unique challenge.

uncommon. Military units often form plans, split up, and execute complicated maneuvers with little to no contact with each other [Dup90].

### ***6.1.2 Homogeneous vs. Heterogeneous Policies***

To investigate whether role-differentiation helps, teams of cooperative agents can be homogeneous or heterogeneous. Homogeneous teams must rely on their current perceived state to differentiate themselves, which is effective in some tasks [BM03]. In contrast, heterogeneous teams have more tactical options available because the search does not need to find one global policy that works for all agents in all cases, that is, it can separate the problem among agents. Such separation can be distributed logically across the team (e.g. agents on the left attack prey on the left). Additionally, while the policies may be heterogeneous, they likely should overlap by a significant amount (e.g. all predators know how to turn to face prey).

In three-predator/one-prey predator-prey, Yong and Miikkulainen [YM07] showed that *coevolved* heterogeneous teams are more effective than homogeneous. This chapter takes this result a step further by testing whether larger heterogeneous teams generated by indirect encodings can encode appropriate patterns of behavior. Thus both homogeneous and heterogeneous teams, as described in Chapter 5, are compared in each experiment.

### 6.1.3 Seeding

In addition to starting evolution from scratch as normal, this experiment also investigates injecting knowledge into the initial search by *seeding* evolution so that the initial population contains variations on a seed genome. While this genome can be from previous evolution or hand-crafted to exploit a particular aspect of the problem, the interesting idea afforded by multiagent HyperNEAT is to seed multiagent learning with the genome of a *single* agent (Section 5.3). Seeding a team with a single strong agent is effective because a single policy with a set of basic skills is faster and easier to evolve than a whole team.

To verify that seeding in this manner is useful in this domain, both heterogeneous and homogeneous teams are tested with and without evolutionary seeds. The seed is created by evolving a single predator agent that is evaluated only on its ability to chase prey, which is a good starting point for multiagent predators. Strong single agents were typically discovered

in less than 50 generations and the best-performing agent among them is the seed for the reported experiments.

#### ***6.1.4 Prey Formations***

Agent teams face two conflicting goals: robust generalization and specialization for efficiency. To balance these goals, while each team is trained on only one of three prey formations (triangle, diamond, and square; figure 6.2), they are trained on two variations of that formation. Training on only one formation encourages discovering specific tactics to deal with the specific formation, enabling the teams to capture prey more quickly. However, training on multiple variations of the same formation encourages teams to develop robustness to minor changes in that formation. To generate these variations, while the number of prey is constant, their locations are changed by varying the angle or length of the formations.

Each formation creates a significantly different challenge because they not only vary in number of prey, but also in how many prey can be initially seen by each predator. Thus each approach is tested on a variety of multiagent scenarios.

## 6.2 Predator-Prey Results

Performance in this section is measured as the time remaining after capturing all the prey, averaged across each formation variant. Each trial is run for 5,000 time steps. The maximum (though impossible) score is thus 5,000 and the minimum score is 0 if all prey are not captured. This measure, which highlights completion of the task, does not necessarily increase with generations like fitness because if a potential solution solves one training example but not the other, it may have to sacrifice performance on the solved example to solve the other. Nevertheless, performance follows a general upward trend over generations and reveals the ultimate quality of solutions.

### 6.2.1 *Training Performance*

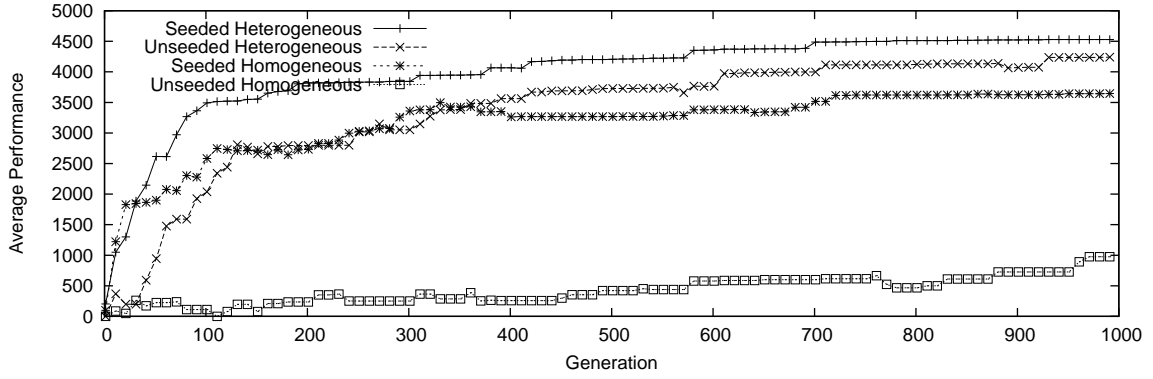
Figure 6.3 shows training performance over generations for teams with homogeneous and heterogeneous policies, with and without seeding, on the formations in figure 6.2. In all three scenarios, the most successful approach was the seeded heterogeneous team, which outperformed all teams across all configurations. Although the difference between seeded heterogeneous and heterogeneous is only significant in the square formation ( $p < 0.001$  after generation 714 according to Student’s t-test) and diamond formation ( $p < 0.05$  after generation 521), it is significant versus both homogeneous approaches on all formations (eventually at least  $p < 0.01$ ). Also in every formation, unseeded heterogeneous performed



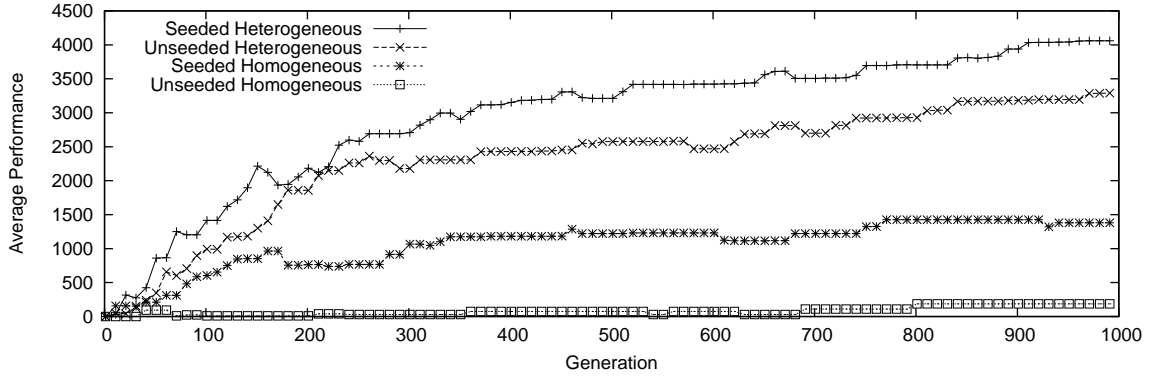
second best (eventually at least  $p < 0.05$  on all formations), followed by seeded homogeneous ( $p < 0.001$  after generation 10 versus homogeneous on triangle, 124 on diamond, and not significant on square) and unseeded homogeneous. Both homogeneous team types could not consistently solve training examples.

A potential question is whether the heterogeneous substrate gains an advantage simply by having more nodes. To check this possibility, all homogeneous experiments were repeated with a substrate with the same number of hidden nodes as the heterogeneous substrate (i.e. a single member of the homogeneous team has 25 nodes, as many as the entire team of heterogeneous agents). The performance of the resulting homogeneous teams was not significantly different, which confirms that the distribution of policies on the substrate is what favors heterogeneous teams.

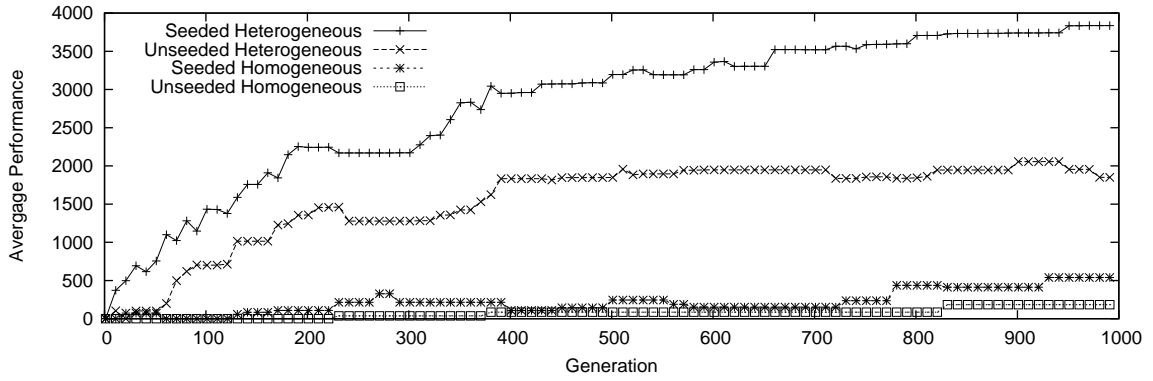
While the different formations do not change the overall ranking of the policy distributions, the teams did perform differently on them. It turns out that the formations increase in difficulty by number of prey because as the number of prey increases, the viability of picking off one prey at a time decreases; thus the teams require more holistic solutions to capture more prey, widening the gap between heterogeneous and homogeneous teams.



(a) Triangle Formation Performance



(b) Diamond Formation Performance



(c) Square Formation Performance

Figure 6.3: Comparing Performance of Different Training Methods. The performance of each training method on the three formations is shown, each averaged over twenty runs. In all cases heterogeneous teams significantly outperform homogeneous teams and seeded teams outperform unseeded.

### 6.2.2 Generalization

Solutions were tested for their ability to generalize to seven variants of each training formation. Diamond formations vary in length from 100 to 300 units, squares vary in side length from 75 to 225 units, and triangles vary from  $0^\circ$  to  $180^\circ$ . The most general solutions perform well on both training *and* testing, for a total of nine variants of each formation. To make the comparison fair, only the most general solutions produced by each of the twenty runs of evolution are compared. That way reported results indicate the best each approach can do. This method of testing generalization follows Gruau et al. [GWP96] and is designed to compare the best overall individuals.

Figure 6.4 shows that generalization performance is highly correlated with training performance (i.e. the ranking of approaches is the same as in training), which means that heterogeneous roles provide a significant advantage. Only the seeded heterogeneous strategy produces teams that could solve all nine scenarios. The most general teams usually employ the same policy for each variant, although some heterogeneous teams change their policy depending on the specific variant.

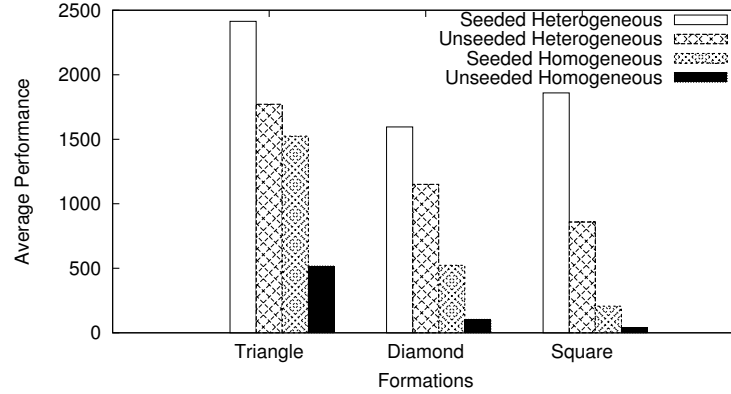


Figure 6.4: Generalization Performance. Average performance is shown for each approach on the nine variants of each prey formation, averaged over twenty runs. Heterogeneous methods generalize significantly better than homogeneous (at least  $p < 0.01$  in all cases except unseeded heterogeneous versus seeded homogeneous on triangle) and seeding produces significantly better performance than starting from scratch (at least  $p < 0.05$  in all cases). The main conclusion is that both heterogeneity and seeding afford significant advantage in generalizing in this domain.

### 6.2.3 Typical Behaviors

This section discusses the different strategies discovered by each of the team representations.

Videos of coordinated team behaviors are available at <http://eplex.cs.ucf.edu/mah.html>

The best pure homogeneous teams rely almost exclusively on every predator finding a prey and chasing it in a circular pattern; if two of these circles overlap one of the predators eventually sees the prey being chased by the other and starts to move toward it. If the predators are well-aligned when one sees the other's prey, they continue to move toward each other and capture both prey; if they are not aligned, one predator abandons its prey to capture the one being chased by the other predator. This policy is somewhat general across prey formations, but if there are no other predators nearby, one predator may chase one prey

forever. Also, it is inefficient because much time is wasted by running in circles rather than capturing prey.

In contrast, heterogeneous agents employ a variety of effective techniques, many exhibiting interesting policy variations based on the team's initial geometric layout, confirming the ability of the multiagent CPPN to encode patterns across the substrate. In one policy, predators hunt for prey in packs of two or three, commonly teaming with adjacent agents. Upon seeing a prey, they approach from opposite sides and either capture the prey with a pincer attack or trap the prey between them, moving in parallel with the prey until they eventually turn to face and capture it. A second heterogeneous policy is *corralling*, in which predators compress the prey into a tightly-packed cluster and then capture them (figure 6.5). A variation of corralling involves some of the predators moving around the perimeter of the prey and forcing them to run toward the center of the formation, while others form a *fence* and slowly advance toward the cluster. Corralling is usually symmetric and a result of policy mirroring, which means that predators from the east and west starting positions rotate around the prey in opposite directions to maximize efficiency. A third policy deploys *posts*, which are usually the predators on the left and right starting positions, who serve as stationary traps. Mobile predators then chase prey into the posts. Another interesting policy involves a single predator (usually the center) advancing forward while other predators chase prey towards it. Almost all heterogeneous solutions rely on one or a combination of these policies, although a few exhibit behaviors similar to homogeneous solutions.

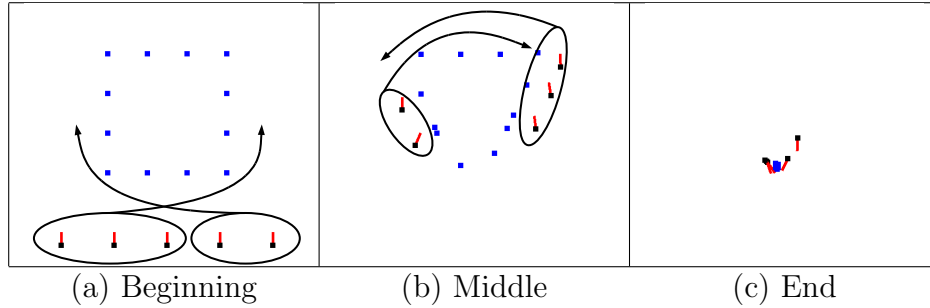


Figure 6.5: Corraling Example. The predators begin by crossing to the opposite side of the prey from which they started (a), causing the prey to retreat inward. The predators then repeatedly circle the prey (b) who continue moving inward away from the predators until they are compacted enough that the predators can easily surround and capture them (c).

While seeded and unseeded solutions employ generally the same tactics, the major difference is that unseeded teams often include agents that serve no useful purpose. Such agents spin in circles, move away from the prey, or just sit still. This inefficiency explains the overall lower performance of unseeded strategies, confirming that starting with basic skills provides a solid foundation for eventual differentiation.

### 6.3 Predator-Prey Discussion

The result that heterogeneous teams significantly outperform homogeneous in training and generalization demonstrates that the multiagent HyperNEAT approach successfully encodes heterogeneous roles that contribute to superior performance. The ability to encode patterns of behavior across a team is critical to success in multiagent learning and thereby addresses a major challenge in the field. The HyperNEAT method allows team behavior to be represented

as variation on a theme encoded in a *single genome*, meaning that key skills need not be rediscovered for separate agents. Furthermore, because multiagent policies are represented by a CPPN, they are assigned to separate agents as a function of their relative geometry, while simultaneously exploiting the agents' internal geometries.

Seeding evolution was also beneficial. This capability captures the idea that real-life teams (e.g. in soccer) often share a critical basic skill set that can be learned faster by an individual agent than an entire team. While HyperNEAT naturally encodes variations on a theme, finding the right underlying theme can initially be challenging. Seeding bootstraps the process, providing a mechanism to inject domain knowledge.

While these results are promising, a further desirable property of multiagent systems is scalability [PL05], and an exciting potential extension to multiagent HyperNEAT is the ability to change the team size *without* further evolution. Because HyperNEAT CPPNs encode policies as a function of their positions, individuals added to the substrate at new positions should naturally receive gracefully interpolated policies. Also, in the same way, the team can dynamically reassign policies when an agent is lost or damaged simply by shifting their positions on the substrate. The next chapter discusses changes to the algorithm and representation that allow such scaling to take place and presents scaling results in predator-prey and room clearing domains.

## CHAPTER 7

### SCALABLE MULTIAGENT HYPERNEAT

As discussed in Section 2.1.1, traditional multiagent learning techniques struggle to represent cooperative heterogeneous teams of more than a few agents and there are typically few rules or principles to determine how *additional* agents could be added after learning has taken place. However, in many tasks, it would be most convenient if the number of possible agents was unbounded and independent of the number initially trained. Whether in search and rescue operations or futuristic nanotechnology procedures, the potential utility of deploying thousands of agents should be achievable through multiagent learning. In fact, in their recent survey of cooperative multiagent learning, Panait and Luke [PL05] cite scalability to be a “major open topic” in the field and go on to say,

The “multi” in multi-agent learning cries out for larger numbers of agents, in the range of ten to thousands or more. Two- and three-agent scenarios are reasonable simplifications to make theoretical analysis feasible: but the experimental and empirical literature ought to strive for more.

Furthermore, because agents may break down or additional ones may become available, ideally the size of a learned team should be dynamically adjustable *after* deployment. While



in the homogeneous case scaling is simply accomplished by adding or subtracting agents with the same control policy, scaling heterogeneous teams is in principle significantly more complicated.

Recall again the soccer team in figure 1.1, which includes eleven agents with assigned roles. How can additional agents be added to such a team, e.g. between the midfielders and the forwards? Intuitively, the implicit policy geometry suggests that these new agents should interpolate between the policies of the surrounding agents, that is, they should be relatively offensive, but not as offensive as the players in front of them. Traditional techniques have no way to exploit this policy geometry because they treat each agent independently, and would thus require retraining to assign such new roles to the new agents. However, because teams in multiagent HyperNEAT are represented by the CPPN as a *pattern of policies* rather than as individual agents, the CPPN effectively encodes an infinite number of heterogeneous agents that can be sampled as needed without the need for additional learning. Thus, if more agents are required, the substrate can in principle be updated to encompass the new agents and resampled to assign policies to them without further evolving the CPPN.

The next section explores an alternative to the substrate layout discussed in Chapter 5, which has properties that make it more amenable to scaling.

## 7.1 Substrate Scaling

In theory the heterogeneous substrate can accommodate a variable number of agents by changing the horizontal ( $x$ ) length of each agent controller so that the desired number of controllers fit in the same space as the original number (figure 7.1). The  $r(x)$  function that produces the repeating coordinate frame for each agent is also changed so that it repeats the appropriate number of times and in the appropriate places, which is accomplished simply by changing its period. Thus even though the agents become smaller with respect to the absolute  $x$  coordinates, they maintain the same internal coordinates because of  $r(x)$ . In this way, interestingly, the policy geometry is exploited to interpolate the roles of new agents. However, as explained next, scaling in this manner may not produce optimal results.

### 7.1.1 *Alternative Substrate*

The danger with scaling the multiagent substrate presented thus far is the confusion that may result from conflating coordinate axes that have different meanings. In particular,  $r(x)$ , which is the internal  $x$ -axis of the agent's *ANN*, is defined as a periodic function of the global  $x$ -axis of the *team*. In addition to conflating these potentially orthogonal axes, to scale the team, the geometric relationships of the nodes on the substrate must be altered. Specifically, the horizontal size of each ANN is reduced, and some of the nodes of new agents end up where the nodes of old agents previously existed. Figure 7.2 demonstrates this conflict by

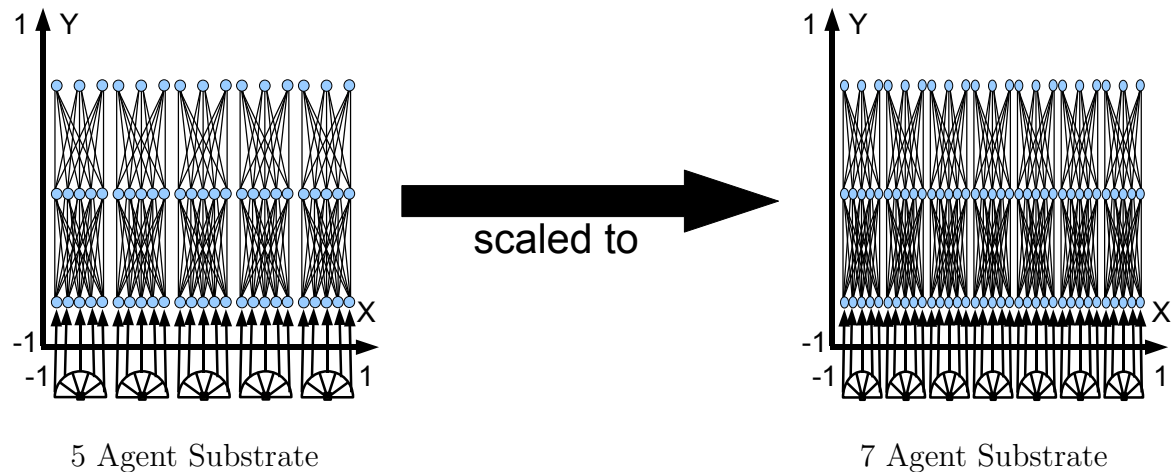


Figure 7.1: Initial Concept for Heterogeneous Scaling. Because multiagent HyperNEAT represents teams as a pattern of policies, it is possible in principle to interpolate new policies in the policy geometry for additional agents by sampling new points on the substrate. The substrate remains the same size and the additional agents are squeezed horizontally so that the new number of agents fit. Additionally, the  $r(x)$  function is altered slightly so that it repeats the correct number of times. However, as explained in Section 7.1.1, this approach to interpolation can still be improved.

showing the boundaries of agents on a team of size five and how those boundaries shift when the team is scaled up to seven agents. The problem is that if the learned policy geometry is based heavily on the global  $x$  coordinates then the pattern could be disrupted across multiple team sizes because the global coordinates may not refer to the same agent on which the pattern was trained.

Thus, as an alternative to the original approach, axes that define coordinates within the agent’s “head” and within the team at large can avoid conflation by being *orthogonal* to each other. Because the ANN already has two axes (assuming it is in two-dimensional space), the geometric configuration that captures the appropriate orthogonal structure is three-dimensional. In this configuration, agent position is now along the newly-introduced

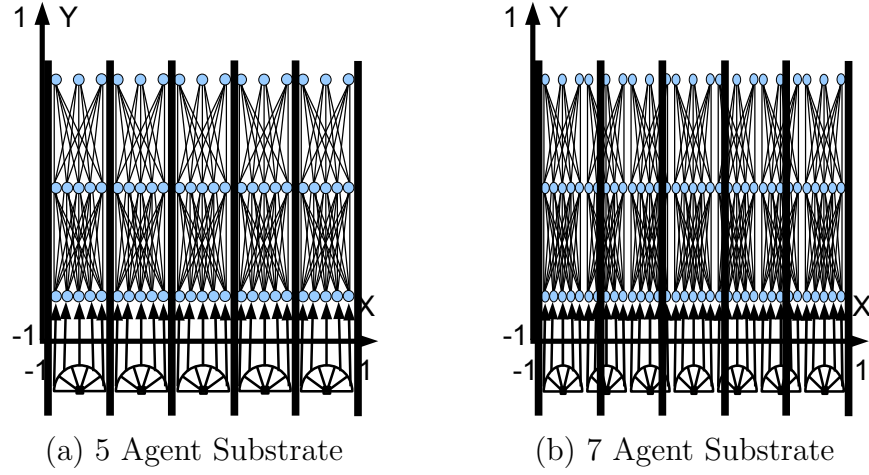


Figure 7.2: Potential Problem with Scaling. The substrate based on the  $r(x)$  function has a fixed amount of space that is evenly divided among all the agents that are on the team. When additional agents are added, the agents are compressed horizontally to accommodate the new number, resulting in parts of the space that used to compose one agent now containing parts of other agents. This problem is illustrated by observing the agents' borders at team sizes five (a) and seven (b). If the pattern of policies is based heavily on the global  $x$  coordinate, new policies may not correctly interpolate because of this conflation.

$z$ -axis;  $r(x)$  is no longer necessary, and  $z$  simply represents the discrete position of the agent on the team. Additionally, because the ANN for each agent exists at a discrete  $z$  coordinate, there is no longer a need to compress the substrate coordinates: There is effectively room for an infinite number of agents without conflict. When scaling, agents can therefore simply be added to the stack at discrete points (figure 7.3). This method is called the *stacked substrate* to differentiate it from the previous substrate, which will be called the *divided substrate*.

It was previously shown that the divided substrate could be seeded with a strong starter policy (Section 5.3). The stacked substrate also possesses this capability. Recall that a CPPN that represents a single agent takes the four inputs  $x_1$ ,  $y_1$ ,  $x_2$ , and  $y_2$  (figure 5.1a). This CPPN represents a two-dimensional connectivity pattern, whereas the stacked agent

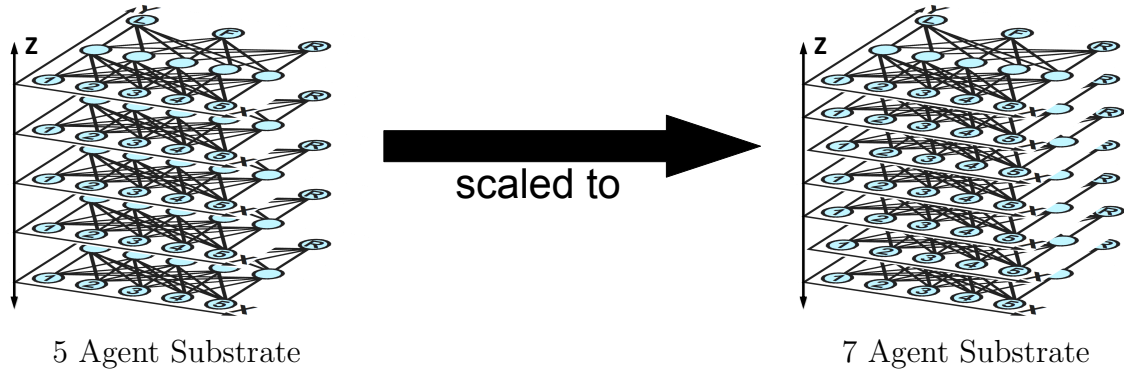


Figure 7.3: Stacked Substrate Heterogeneous Scaling. The *stacked substrate* places the ANN for each agent at a coordinate on the  $z$ -axis, effectively making a stack of two-dimensional substrates. It is scaled by inserting new two-dimensional substrate slices along the  $z$ -axis. Because each new slice exists at a fixed  $z$  coordinate, adding new agents does not affect existing agents. This representation is thus more amenable to scaling than the divided approach (figure 7.2).

substrate is three-dimensional. Therefore, a new  $z$  input is added to the network, although with no connections to the existing network (figure 7.4). Only one  $z$  input is necessary because each agent exists at an infinitesimal point on the  $z$ -axis. In this way, the CPPN can now be queried for the policies of a *team* of agents. However, because the only factor that differentiates the agents,  $z$ , is not connected to the network, the team is initially homogeneous (as with the initial seeded divided substrate). However, once  $z$  is connected to the network by mutation, the CPPN can create variations of the seed policy based on the policy geometry along  $z$ .

To demonstrate the benefit of the stacked substrate over the divided substrate, it is necessary to compare the two to ensure that the purported benefits of the stack do not come at the price of representational power or hidden disadvantages. Therefore, both substrates are next tested and compared in the predator-prey domain from Chapter 6.

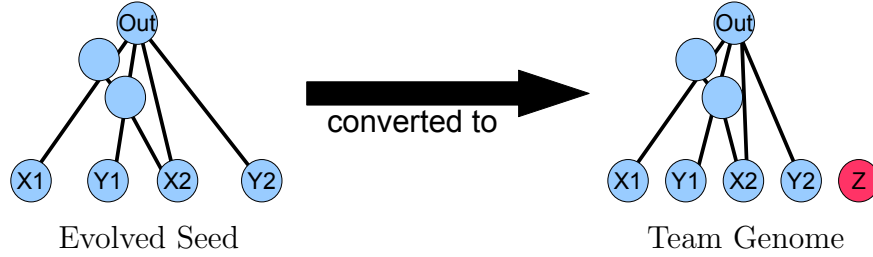


Figure 7.4: Stacked Substrate Seeding. The stacked substrate can also be seeded with a high-performing single-agent policy. To do so, the original CPPN (left) is given a new  $z$  input that determines which agent is being sampled. This method preserves the original seed pattern, but again allows multiagent HyperNEAT to create a pattern of policies relevant to the team’s policy geometry, which varies along  $z$ .

### 7.1.2 Substrate Validation Experiment

The setup for this experiment is exactly the same as the original predator-prey experiment so as to compare the two substrates fairly. However, to account for minor code changes after the original experiment, new divided and homogeneous teams were trained along with the stacked teams.

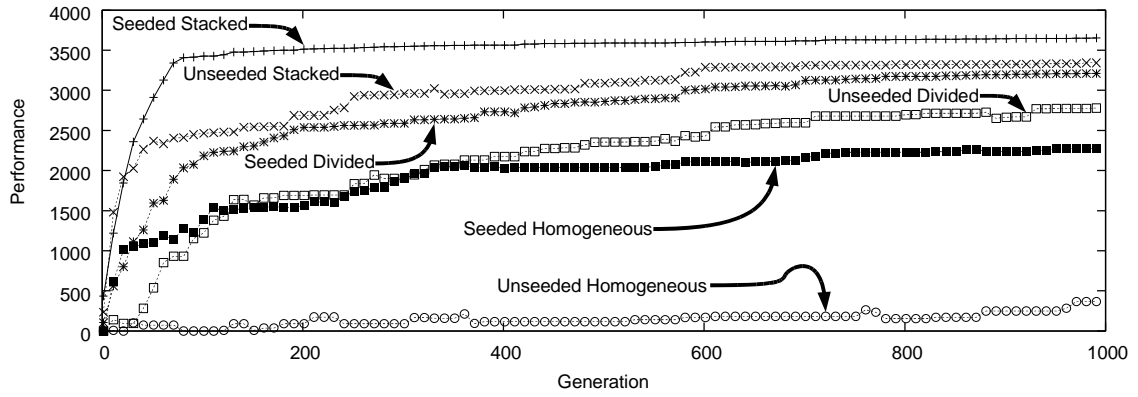
Figure 7.5 shows the training performance of the teams on the three symmetric formations (figure 6.2). In all cases, the seeded stacked teams find the most efficient policies and do so significantly more quickly than the divided substrate and homogeneous teams (eventually at least  $p < 0.01$  in all cases). The seeded stack was also the only team encoding that found a successful solution in every run of all three formations. Heterogeneous teams, regardless of substrate and seeding, consistently significantly outperformed both seeded and unseeded homogeneous teams (eventually at least  $p < 0.01$  in all cases), again confirming the utility of a team-wide policy geometry. Additionally, in all but a few heterogeneous cases (triangle

stacked and triangle divided) the seeded teams significantly outperformed their unseeded counterparts (eventually at least  $p < 0.05$  in all such cases). Thus the teams still exploit the fact that good solutions exist close to pure homogeneity on the continuum of heterogeneity.

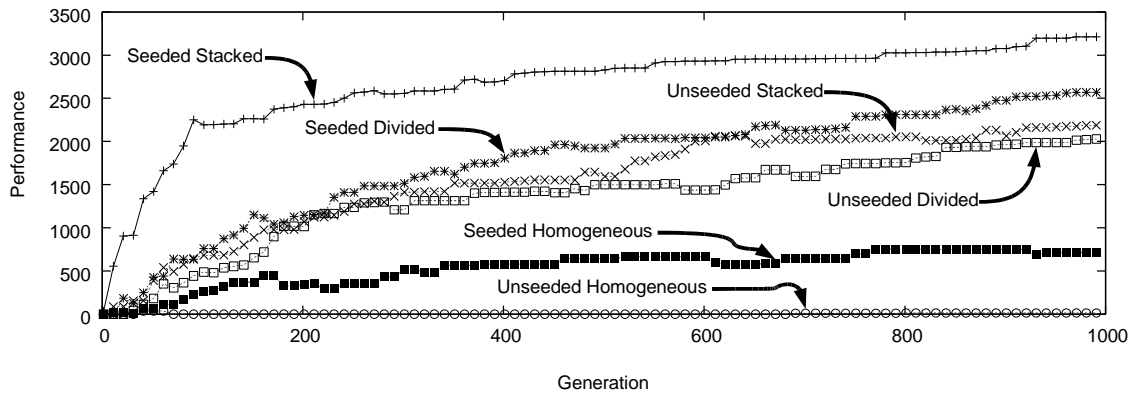
Most importantly, in all cases, the stacked substrate outperformed the divided substrate, suggesting that it is at least as powerful for representing heterogeneous multiagent teams. However, the main reason for introducing the stacked substrate was to facilitate scaling teams to larger sizes, which is the focus of the next section.

## 7.2 Scaling Experiments

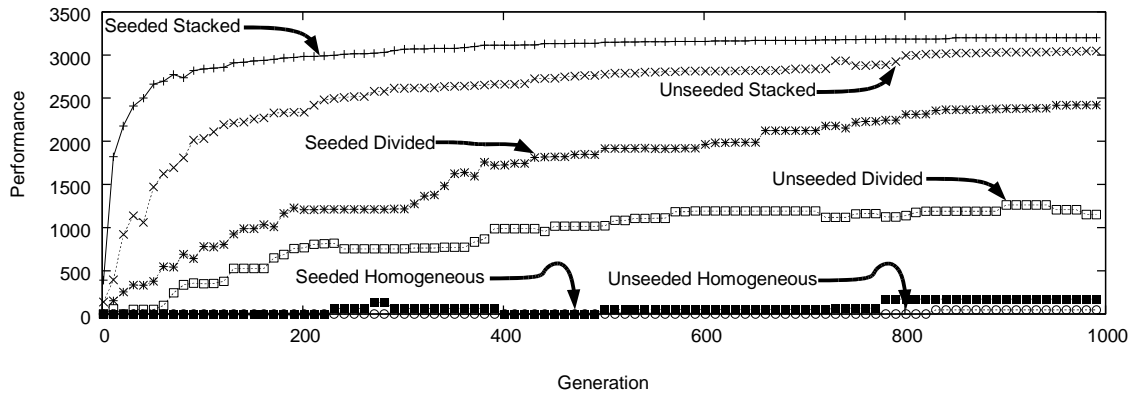
To scale without further training, the algorithm must have an intelligent way to generate new policies for new agents. The previous section described such a method for two different substrates. The experiment described next expands on the results from the previous section by applying the scaling approaches to the solutions found in the predator-prey domain. This section then describes another multiagent scaling experiment in a room clearing domain, which further tests the algorithm’s ability to scale, but in a more real-world application.



(a) Performance for Triangle Training



(b) Performance for Diamond Training



(c) Performance for Square Training

Figure 7.5: Training Performance of Different Substrates. The training performance of the various teams is shown for each of the three formation shapes, averaged over twenty runs. In all cases, the seeded stacked substrate teams learned the best solutions the fastest. Additionally, heterogeneous teams consistently outperformed pure homogeneous teams and seeded teams consistently outperformed their unseeded counterparts.



### 7.2.1 Predator-Prey Scaling Performance

Because multiagent HyperNEAT encodes a team as a pattern of policies distributed across the geometry of the team rather than as a group of individual agents, each CPPN can encode the policies of any number of agents, allowing the team size to change *without further training*. To confirm this capability, the same teams from Section 7.1.2 that were originally trained with only five agents were further tested at team sizes of 7, 9, 25, 100, and 1,000 agents on the same three prey formations. The policies of new agents in heterogeneous configurations are determined by inserting new networks into the substrate as in figure 7.1 and figure 7.3. The main question is whether performance can be maintained (and even improved) after scaling, and which substrate is most effective.

Figure 7.6 and figure 7.7 show the number of successes out of 40 (20 runs on two formation variations) and the time taken to complete the task for these new team sizes, respectively. Success is defined as capturing all the prey within the time limit and speed is measured by how much time is taken to capture all the prey (the full 2,000 ticks per task is given if the team is unable to capture all of them). The data was collected by testing every generation champion from each of the 20 runs on all scaling sizes. To compare the best overall individuals, the one that most quickly captures all the prey in the most new scenarios is chosen to represent each run. For example, a team that captures all prey at sizes 7, 9, 25 and 100 and takes 500 time steps at each would be preferable to a team that captures all prey at only 100 and 1,000 regardless of speed. It would also be preferable to a team that

also captures all prey at 7, 9, 25 and 100 but takes 750 for each. For each type of substrate, the results of the best scaler from each of the 20 runs is averaged. This method of testing follows from Gruau et al. [GWP96] and is designed to compare the best overall individuals. Its advantage is that it gives a sense of the best that is possible to achieve from each method.

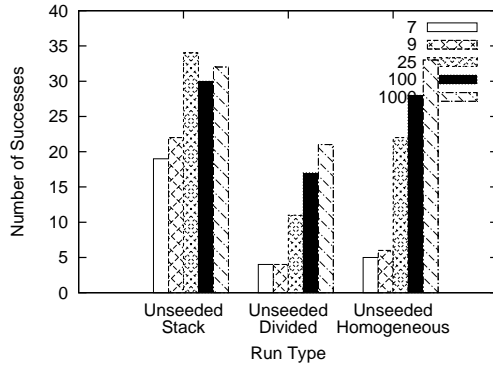
This measure of performance is also similar to the jump-start metric in transfer learning [DAR05], in which policies are evaluated based on the initial performance boost (typically in the first few learning iterations) when transferred to a new domain, as compared to starting from scratch in that domain. The difference from the treatment in this chapter is that in transfer learning the policies typically undergo further training, whereas the goal for multiagent HyperNEAT in this experiment is to be able to create policies that can be effective immediately and *without further learning* in a larger team size.

Note that in general as team size increases, performance should improve because more predators are available against the same number of prey. However, for that to happen, agents must remain coordinated even as more are added because the agents cannot sense each other and therefore it is possible for new agents to unknowingly interfere or simply be redundant with the strategies of old agents. Thus the question is whether such coordination is indeed maintained, leading to a gain in performance from adding more agents, and which method is most effective at exploiting the ability to add more agents overall.

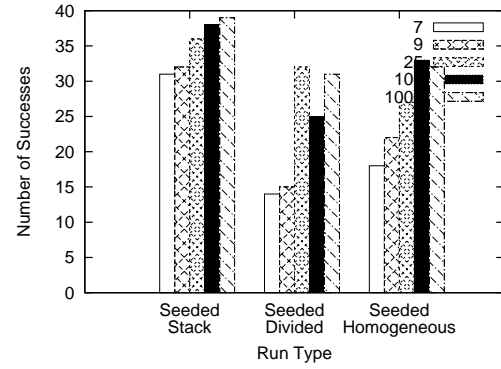
The main result is that multiagent HyperNEAT makes it possible for teams to increase in size without additional learning. Scaling performance generally mirrors training performance in the sense that the stacked substrates significantly outperform (both in terms of success rate

and speed) the divided substrates (at least  $p < 0.01$ ) except for seeded diamond at size 100 and unseeded square sizes 100 and 1,000. Divided substrates in turn outperform the purely homogeneous teams (at least  $p < 0.01$ ) except in the triangle cases, in which the simplicity of the formation allowed homogeneous scaling to be more effective (although the stacked substrate is still generally superior in these cases). Additionally, the seeded teams significantly outperform the unseeded versions in the majority of cases (at least  $p < 0.05$ ). However, there are some exceptions. For example, in the triangle formation, the homogeneous teams perform competitively when scaled to large team sizes (figure 7.6a and figure 7.6b). This boost in performance is explained by the typical behavior of homogeneous agents, which is to target the closest agent and hope for a collision with another predator. By increasing the number of predators and leaving the number of prey the same, the chance for a collision increases. However, while they can work, such strategies are still inefficient: Note the difference in the significantly slower speed of the homogeneous teams in figure 7.7a and figure 7.7b and the overall poorer performance on the other formations (figure 7.6c-f). For videos of scaled agent teams see: <http://eplex.cs.ucf.edu/multiagent-hyperneat-scaling.html>.

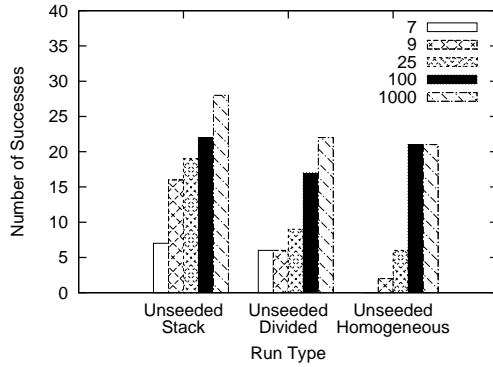
Thus the main conclusion is that heterogeneous teams (particularly from stacked substrates), which can interpolate new agent roles through the policy geometry, scale best (particularly in terms of task completion speed). The superior performance of the stacked substrate in both training and scaling further suggests that separating individual geometry from policy geometry is the best approach to configuring the substrate. Therefore, future



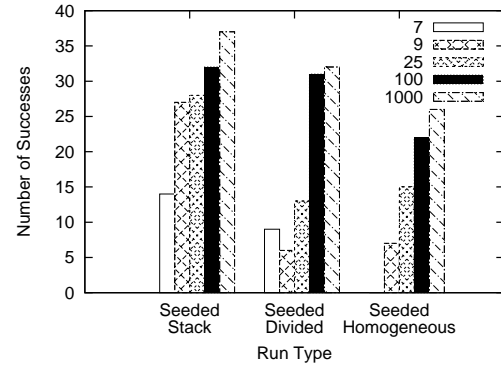
(a) Unseeded Trained on Triangle



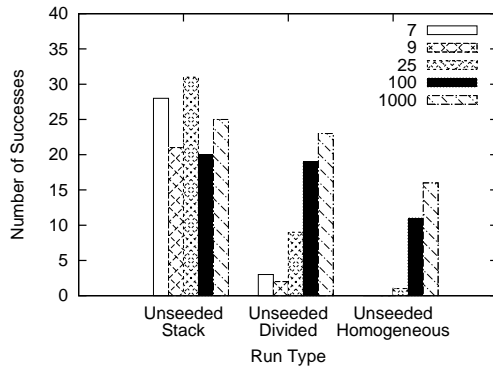
(b) Seeded Trained on Triangle



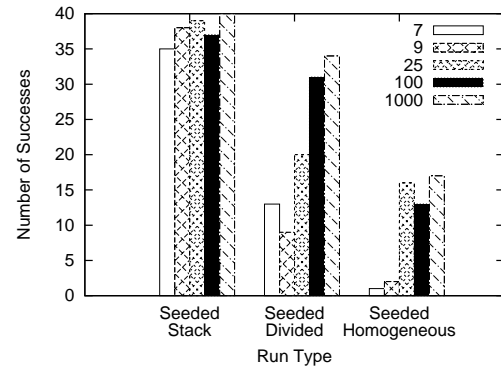
(c) Unseeded Trained on Diamond



(d) Seeded Trained on Diamond



(e) Unseeded Trained on Square



(f) Seeded Trained on Square

Figure 7.6: Scaling Success Rates of Different Substrates (lower is better). The number of trials in which all prey were captured is shown for each of the three formation shapes, summed over twenty runs, with two trials per run. Success rates were determined by the averaging the performance of the team from each run with the best performance on all team sizes over all twenty runs. Scaling performance generally mirrors training performance: Seeded stacked is the best overall, heterogeneous is better than pure homogeneous, and seeded is better than unseeded.

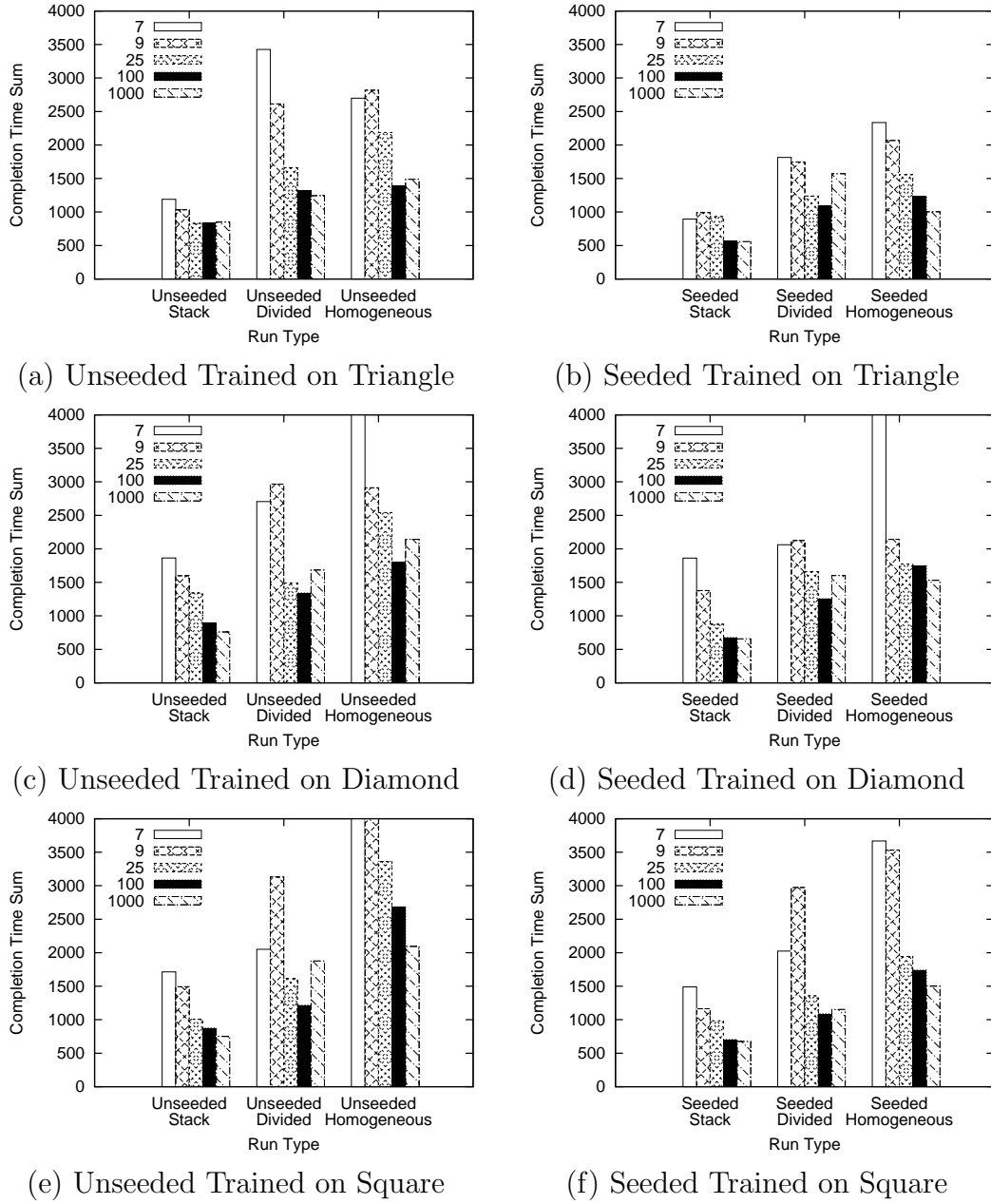
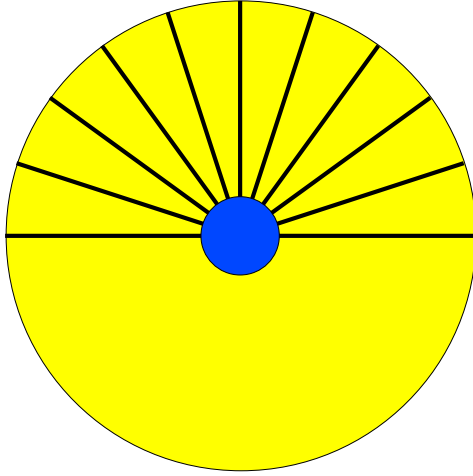


Figure 7.7: Scaling Performance of Different Substrates (lower is better). The average amount of time taken out of 2,000, summed over both formations, is shown for each team composition averaged over 20 runs on various team sizes on the three formations. A score of 4,000 means no team was able to complete that size. Scaling performance is determined by averaging the performance of the team from each run with the best performance on all team sizes over all twenty runs. Again, seeded stacked is the best overall, heterogeneous is better than pure homogeneous, and seeded is better than unseeded.

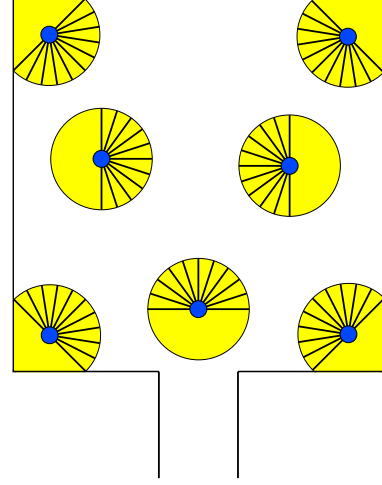
experiments will abandon the divided substrate in favor of the stacked, as in the next section, where multiagent HyperNEAT is shown to scale in a different multiagent task.

### 7.2.2 *Room Clearing Experiment*

In the room-clearing domain, a group of agents enter a room and position themselves to visually cover as much of the room as possible (assuming 360 degree rotating cameras). Room clearing is a good domain to further demonstrate the abilities of multiagent HyperNEAT because in this domain, sensory information alone is inadequate to robustly solve the task and physical collisions between agents are enforced even though they cannot see each other. It is also an important military task that may one day be performed by unmanned ground vehicles. The optimal solution is for each agent to be as far as possible from other agents. However, the agents' rangefinder sensors do not directly specify whether a particular area is already covered by another agent; thus, *a priori* role assignment is critical to obtaining good performance. Coordination among varied policies is also necessary because colliding with other unseen agents may prevent an agent from reaching its desired destination. The ability to avoid walls and navigate the room are also important for all the agents. As in the predator-prey domain, multiagent HyperNEAT has the ability to balance the need for these heterogeneous and homogeneous elements of policy across teams with any number of



(a) Room Clearing Robot



(b) Room Clearing Domain

Figure 7.8: Room Clearing Domain. The agents in the room clearing domain (a) are represented by the small blue circle. They have 11 rangefinder sensors denoted by the rays emanating from the agent and have a visual range depicted by the large yellow circle. The agents must enter a room and spread out so that their combined visual ranges cover as much area as possible (b).

agents. Based on the predator-prey results, only the stacked substrate is tested against a homogeneous team in this experiment.

Agents are controlled by ANNs similar in architecture to those in the predator-prey domain, but augmented with recurrent connections so that agents can potentially retain a basic memory of past states. Each agent (figure 7.8a) has 11 rangefinder sensors that indicate the distance to nearby walls but not nearby agents, which reflects a possible configuration of sensors on real robots and also makes the task more difficult. On each discrete time step, an agent's ANN outputs dictate whether it turns and/or moves in small increments. If an agent requests to move at a velocity that is below a certain threshold, it is interpreted as a request to stop, and the agent's motor is disabled for the remainder of the evaluation. The

room-clearing teams are trained at a size of seven agents because preliminary experiments showed that seven is the smallest team size for which avoiding overlap is non-trivial. Thus, because the principle of spreading is the key to scaling effectively, seven is a good size on which to start to produce a team optimized to spread.

Agents are evaluated in a single rectangular room (figure 7.8b) for 35 simulated seconds comprised of 0.2 second discrete time steps. Agents begin outside the room in an evenly spaced vertical line; a fixed policy of moving directly forward is maintained for each agent until they are inside the room, at which point each agent is under control of its evolved ANN for the remainder of the evaluation. To measure how much of the room is covered by the team of agents, a grid is superimposed upon the room. A grid square is deemed covered if an agent is in close proximity to it. For each of the last 15 seconds of the evaluation, the number of grid squares inside the room covered by the agents is counted to determine the team’s fitness score. This measure of fitness encourages agents to spread quickly throughout a room just as a real team of humans would, to rapidly assess risk.

Experimental parameters are in the Appendix.

### 7.2.2.1 Scaling in Room Clearing

To test the scaling capability of multiagent HyperNEAT in this domain, the best CPPN of each generation of training is again tested on several different team sizes *without further*

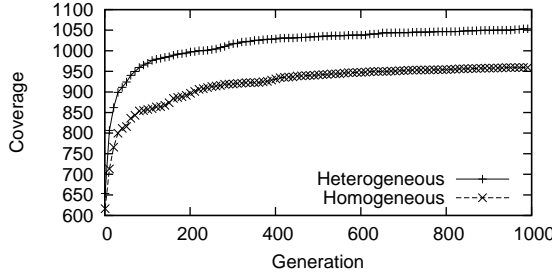


*learning*. The methodology for scaling is the same, but the details of adding new agents to each domain varies slightly. While the teams in predator-prey were trained on a number of sizes that varied substantially (i.e. between 7 and 1,000 agents), the room in the room-clearing domain is too small to accommodate scaling to very large team sizes. Thus these teams tested on all odd-numbered team sizes from 9 to 23. These numbers are also more realistic for an actual military scenario in which some new agents are available to add to a team. That is, teams in the real world generally fluctuate by a few agents rather than orders of magnitude.

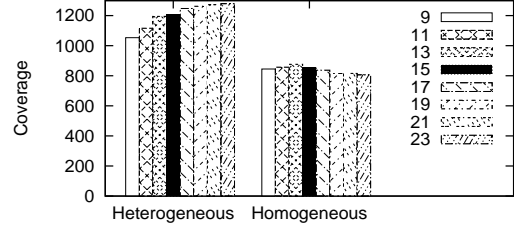
### 7.2.2.2 Room Clearing Results

Figure 7.9a shows the training performance over generations of the teams over 20 runs in the room clearing domain. Results are measured as room coverage over the last 15 seconds (higher is better). Again, heterogeneous teams find significantly more efficient policies and do so more quickly than the homogeneous teams ( $p < 0.001$ ), confirming the utility of a team-wide policy geometry.

Figure 7.9b shows the scaling performance of the the teams, which were derived in the same way as the scaling results from predator-prey. As with predator-prey, as team size increases, performance should generally improve because there are more agents to cover the same size room. However, such improvement is not guaranteed because the agents



(a) Room Clearing Training



(b) Room Clearing Scaling

Figure 7.9: Room Clearing Results. In the room-clearing domain performance (defined as sum of the number of grid squares covered by the team during the evaluation period) is averaged over twenty runs. Heterogeneous teams learned the best solutions and learned them more quickly than homogeneous. Additionally, heterogeneous teams were better able to scale at all teams sizes. In fact homogeneous teams actually begin to lose performance as team size grows due to inefficiencies with their strategies.

cannot sense each other so it is possible for new agents to unknowingly interfere or become redundant with the previously existing agents. Thus the question is whether coordination is maintained so that performance improves as more agents are added. Scaling performance generally mirrors training performance in the sense that the heterogeneous teams significantly outperform ( $p < 0.001$ ) the homogeneous teams. Interestingly, while homogeneous teams also benefit from more agents in the predator-prey task, their performance begins to *decline* as more agents are added in room clearing. For videos of room clearing behaviors see <http://eplex.cs.ucf.edu/mahnaamas2010.html>.

### 7.3 Scaling Discussion

In their recent survey of cooperative multiagent learning, Panait and Luke [PL05] cite scalability to be a “major open topic” in the field. In a first step in this direction, multiagent HyperNEAT was able to create teams of heterogeneous agents that could scale several orders of magnitude in size without further training. Such a task is prohibitive for traditional heterogeneous multiagent learning techniques, yet multiagent HyperNEAT accomplished it by representing the teams as patterns rather than as individual agents and exploiting the policy geometry of the teams.

Scaling existing teams is beneficial for a number of reasons. As discussed earlier, an ideal application of scaling would be to add seamlessly new agents to a team as they become available. Scaling could also be used in the opposite manner, that is, when agents must be removed from a team. For example, they might be malfunctioning, damaged, or just needed elsewhere. In such cases, a team could be dynamically scaled down and continue to operate.

An important difference between the results in predator-prey and in room clearing is that while the heterogeneous policies improve on average in both domains with more agents, the homogeneous policies *degrade* on average as the team gets larger in room clearing. The technical reason for this disparity is that the collisions in the small room mean that adding more agents that all do the same thing will eventually cause a pile-up. However, more generally, it shows that adding more agents is not always automatically helpful; thus the ability to intelligently interpolate intermediate policies along a cooperative spectrum will sometimes

be critical to enabling scaling, highlighting the utility of policy geometry. Furthermore, even though the homogeneous team does improve with more agents in predator-prey, its performance is still significantly worse than every heterogeneous team size, which means that simply improving through scaling is not enough; the idea is to improve a policy that is already good, which is possible through heterogeneous policy geometry.

Finally, this chapter compared two methods of representing teams in multiagent HyperNEAT: the divided substrate and the stacked substrate. In both training and scaling the stacked substrate was significantly better than the divided. This disparity can be largely attributed to the fact that the stacked substrate does not conflate the axis of policy geometry and an axis of the agent’s brain as the divided substrate does. While this conflation may seem to be appropriate, especially when the agents are literally placed in a line such that the geometry of the agent’s brain and the policy geometry are along the same axis (as in the predator-prey experiments), the results show that separating the two, as with the stacked substrate, produces superior results. As a result, experiments in later chapters exclusively use the stacked substrate.

The next chapter revisits the predator-prey domain to compare multiagent HyperNEAT with a multiagent reinforcement learning technique based on SARSA( $\lambda$ ) in both training and scaling large multiagent teams.

## CHAPTER 8

### COMPARISON WITH REINFORCEMENT LEARNING

This chapter serves two important purposes. First, it validates the power of exploiting policy geometry by comparing multiagent HyperNEAT to a multiagent approach that does not exploit it called SARSA( $\lambda$ ) [SB98] in a predator-prey domain. SARSA( $\lambda$ ) is a good choice for this comparison because it is a popular reinforcement learning algorithm that has been applied to both cooperative multiagent systems [SSK05] and predator-prey scenarios [ISK03, KHB05]. Because both multiagent HyperNEAT and SARSA( $\lambda$ ) have been shown to succeed in this type of domains, it makes a good benchmark in which to compare performance. The second purpose of this chapter is to further determine the scalability of multiagent HyperNEAT by both increasing the difficulty of problems when scaling up, and testing the algorithm’s ability to scale *pre-training*, that is, its ability to train very large teams.

#### 8.1 Predator-Prey Experiment

This experiment also takes place in a predator-prey domain similar to the one in Chapters 6 and 7. Recall that the goal is for a team of predators to capture (i.e. intercept) the prey,

but that the predators cannot see each other or communicate, creating a difficult scenario wherein agents must learn complementary policies despite their inability to interact directly. The predators, in fact, must work together because a single predator cannot capture a prey, which runs away from the nearest predator. Thus the prey cannot simply be chased and it is easy for predators to interfere with each other. Unlike other predator-prey domains [JG00, ISK03, KHB05, PT09] the environment is neither a fixed size nor toroidal, allowing an arbitrary number of predators and prey to fit in the world, and facilitating the scaling experiments. However, the predator-prey domain in this chapter also differs in important ways from that presented previously in this dissertation. The remainder of this section details these differences.

### ***8.1.1 Predators and Prey***

The major difference between the predators in this experiment and those presented previously is that, to make the domain more amenable to reinforcement learning (i.e. the SARSA( $\lambda$ ) method), they are limited to a set of discrete actions. That is, rather than choosing a continuous forward velocity and turn angle at each time step, predators instead choose among three set actions: turning left  $90^\circ$ , turning right  $90^\circ$ , or moving forward one unit. The substrate that controls each agent remains unchanged (figure 6.1b), but the decision on which action is performed is based on the highest output, with ties defaulting to forward.

The only difference for the prey agents is that they now move at twice the speed of a predator, but they still run away at the exact opposite angle as the closest predator.

To encode the continuous values returned by the predator’s five rangefinder sensors, SARSA( $\lambda$ ) employs Sutton’s tile coding algorithm [Sut09], with eight tilings and a tile width of 0.2 to create the state representation. Tile coding is a coarse coding method used to increase generalization and encode large or continuous state spaces compactly for reinforcement learning, and has previously been applied to MARL [WB10, SSK05]. The state space is partitioned into several tilings, which are further broken into tiles, such that each tile represents a discrete feature. The number of tilings and tile width were chosen to encourage generalization, and because the values produced the best results in preliminary experiments. For a complete description of tile coding see Sutton [Sut96]. Reinforcement learning agents select actions with an  $\epsilon$ -greedy approach during training, where  $\epsilon = 0.01$ , and pure-greedy selection during testing.

### 8.1.2 *Prey Formations*

The predator team starts each trial in a line, 4 units apart, facing the prey (figure 8.1). The environment the agents inhabit is physically unbounded, and each trial lasts 500 time steps. For SARSA( $\lambda$ ) teams, each agent receives the same reward  $r$  at each time step, where  $r = \frac{p_c}{p_t} - 1$ ,  $p_c$  is the number of prey captured, and  $p_t$  is the total number of prey, until either

the time limit expires or all prey have been captured. This reward scheme follows precedent in prior applications of MARL that reward all agents collectively for team success [SSK05]. Trials were also attempted with individual rewards, but performance was markedly worse, further supporting the chosen scheme. Because they are evolved, HyperNEAT teams do not receive rewards over the trial; instead they are given a fitness value of  $(1,000\frac{p_c}{p_t}) + (500 - t)$ , where  $t$  is the time taken to capture all the prey or 500 if not all prey were captured. Both measures incentivize the predators to capture as many prey as possible, as quickly as possible. Teams are trained on one of two formations: the line or the L (figure 8.1a and b, respectively), both of which presents a different challenge to the teams. Note that a major difference between these formations and those in the previous predator-prey experiments is that the number of prey in a formation is a function of the number of predators that are attempting to capture them. Thus problem difficulty now scales with team size, creating a greater challenge for teams that scale.

In the line, there are half as many prey as predators. The prey are positioned 10 units away from the predators vertically and spaced  $\frac{4(2p_t-1)}{p_t}$  units apart horizontally, starting from the same x-coordinate as the predators, but shifted by half the normal spacing amount. Thus the prey are spread across the same distance as the predators, but with a slight space on the edges to encourage predators to approach the prey. An interesting property of the line regarding cooperation is that it is symmetric, so teams should in principle be able to develop symmetric strategies to capture the prey. In addition, all the prey are seen by at least one predator at the start of the simulation so exploration is not required, that is,



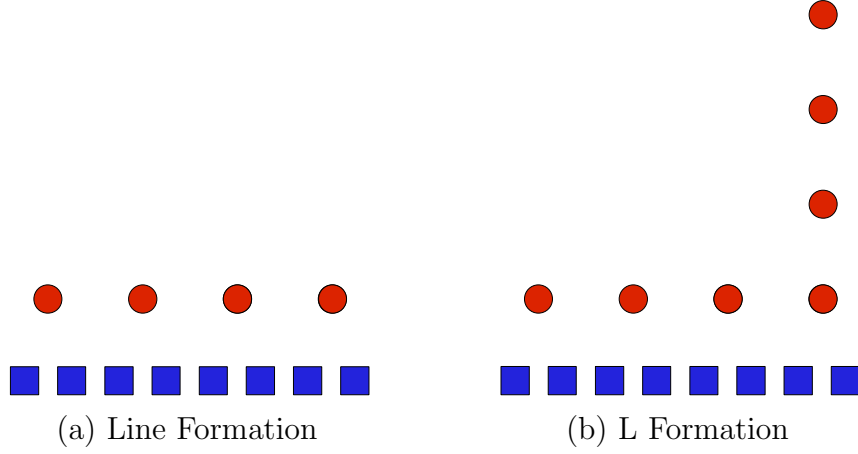


Figure 8.1: Prey Formations. The prey (red circles) are arranged in either a line (a) or an L (b) formation, thereby testing both symmetric and asymmetric scenarios. This figure depicts both formations with eight predators (blue squares), although the number of predators and prey can change. The predators are always placed in an evenly-spaced line below the prey.

together the predators have complete information about the prey. However, because they cannot communicate, individual predators still have limited state information.

The L formation is a more complex test of coordination for the teams because it is asymmetric. Thus the agents have to learn specific roles for specific locations. In previous chapters multiagent HyperNEAT was only tested on symmetric prey formations, but teams were able to develop both symmetric and asymmetric strategies. Also, the prey are not fully-observable from the start, that is, many of them cannot be seen by any predators when the simulation begins. Thus exploration is necessary to solve the task. Finally, the L has almost double the prey of the line, which makes the task more complex and time-consuming. The first half of the prey are placed exactly as in the line formation. The remaining prey are placed behind the left-most prey in a straight line back, five units apart. Note that unlike the line in the L formation, the maximum distance between the a predator and prey increases

depending on the number of prey, thus in sizes 128 and 256 of the L a predator cannot actually reach all the prey within the time limit, so these sizes will not be investigated. Also, size two of the L is identical to the line at size two, so it will also not be tested in these experiments.

### 8.1.3 *Scaling*

In contrast to the scaling experiments in Chapter 7, this chapter discusses two forms of scalability. The first is *pre-training* scaling, wherein the learning algorithm knows how many agents are on the team before training starts. Thus this type of scaling tests the scalability of the learning algorithm itself. Such a property is important if large-scale, real world problems are to be solved with the method. The other type of scaling is scaling *without further learning* as in Chapter 7 and is referred to as *post-training* scaling in this chapter to distinguish the two measures. However, *interpolated scaling* is a unique capability of HyperNEAT, making a direct post-training comparison between methods difficult. Therefore, because SARSA( $\lambda$ ) lacks such a capability, its post-training scaling will be tested by duplicating the policies of existing agents such that when a team is scaled up by a factor of  $S$ , the first  $S$  agents will have the first agent's policy, the second  $S$  agents will have the second agent's policy, and so on.

Thus, both methods will be tested in their ability to scale both pre and post-training with team sizes ranging from 2 – 1,024 agents, wherein each team size is double the last (i.e. 2, 4, 8, etc.). Teams will be trained on sizes of up to 256 agents for the line and 64 for the L and tested on up to 1,024 for the line and 64 for the L.

#### 8.1.4 *Seeding*

Seeding for multiagent HyperNEAT is handled the same way as in previous predator-prey experiments (Section 7.1.1). However, SARSA( $\lambda$ ) can also take advantage of seeding by starting each agent with a known good policy. Therefore, for SARSA( $\lambda$ ), the weights of a single agent are copied to all agents on the team. In both cases the single seed agent was trained to chase prey as closely as possible (recall that a single predator cannot capture prey, so chasing is the best starting point possible). Solutions were found in both cases in under 5,000 evaluations and form the basis for seeding experiments.

Although the initial seeded team for both methods (i.e. a homogeneous group of predators that can chase prey) is unlikely to be able to solve the problem directly, such a team should provide a good starting point for agents to differentiate and then solve the problem. The key difference between SARSA( $\lambda$ ) and multiagent HyperNEAT in this respect is that multiagent HyperNEAT can discover a policy geometry with which to vary this base policy, whereas SARSA( $\lambda$ ) must independently learn how to change each agent’s individual behavior to best

suit the team’s goal. To verify that this capability is important, multiagent HyperNEAT and SARSA( $\lambda$ ) are also tested with seeded policies.

### ***8.1.5 Reinforcement Learning Parameters***

In the SARSA( $\lambda$ ) runs,  $\lambda = 0.9$ ,  $\epsilon = 0.01$ , and  $\alpha = 0.05$ . In addition, the implementation uses Sutton’s tile coding software [Sut09] with five variables, eight tilings, and a tile width of 0.2. These values were found to produce the best results through preliminary experimentation. For multiagent HyperNEAT parameters see the Appendix.

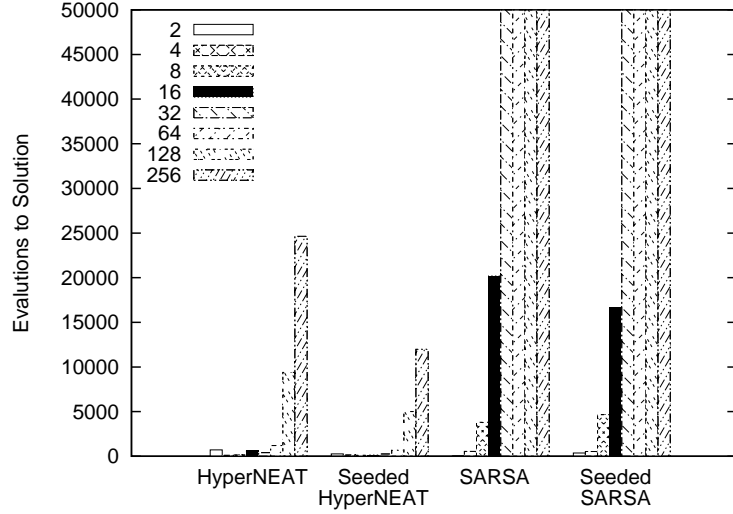
## **8.2 Comparison Results**

To test the pre-training ability of each method to scale, teams were trained on sizes 2, 4, 8, 16, 32, 64, 128, and 256 for the line and 4, 8, 16, 32, and 64 for the L. The main question is whether the methods can continue to find effective solutions as team size (and also the number of prey) increases. Note that because the states of the agents are egocentric and the agents do not see each other, the state-space does not necessarily grow for *individual* agents as in other multiagent problems. However, as more prey are added, the possibility of more sensors being simultaneously activated at different values does increase, meaning that an agent is more likely to encounter a larger number of more varied states as the number

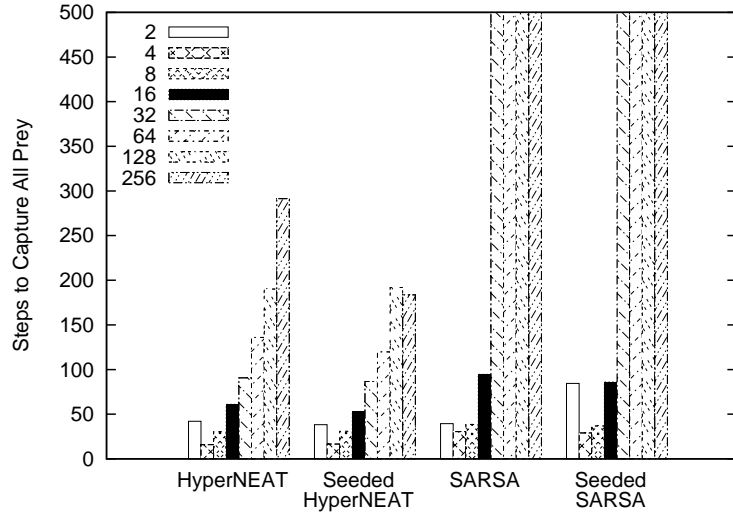
of prey increase. Additionally, as more agents are added, the potential for conflicts among agent policies increases. Figure 8.2 and figure 8.3 show how each method scales with pre-training (i.e. the teams were trained at the desired team size) on the line and L formations, respectively. In both figures, the first graph shows the average number of evaluations until the first solution was found and the second graph shows the average number of timesteps during simulation until all the prey are captured for the best solution of each run. In both cases lower values are better and if a bar is at the maximum value (50,000 evaluations or 500 timesteps), it means that no teams were able to solve the problem at that size for that method.

For the line formation, the most striking result is that while HyperNEAT is able to consistently find solutions for teams of up to 256 agents, SARSA( $\lambda$ ) stops being able to find solutions above only 16 agents. In fact, each time the number of predators and prey doubles, the number of evaluations SARSA( $\lambda$ ) requires to solve the problem increases by an order of magnitude. The results are similar in the L: Multiagent HyperNEAT finds solutions for all tested sizes, but SARSA( $\lambda$ ) can only find solutions for size four. Therefore, multiagent HyperNEAT, with the ability to encode and learn the policies of an entire team and the relationships among them simultaneously, is better able to deal with the conflicts that arise in a large multiagent domain.

Note that the fact that HyperNEAT and SARSA( $\lambda$ ) are comparable up to about eight agents on the line is an important validation that the SARSA( $\lambda$ ) implementation works. Of course, an important concern in comparisons between different approaches is that all are

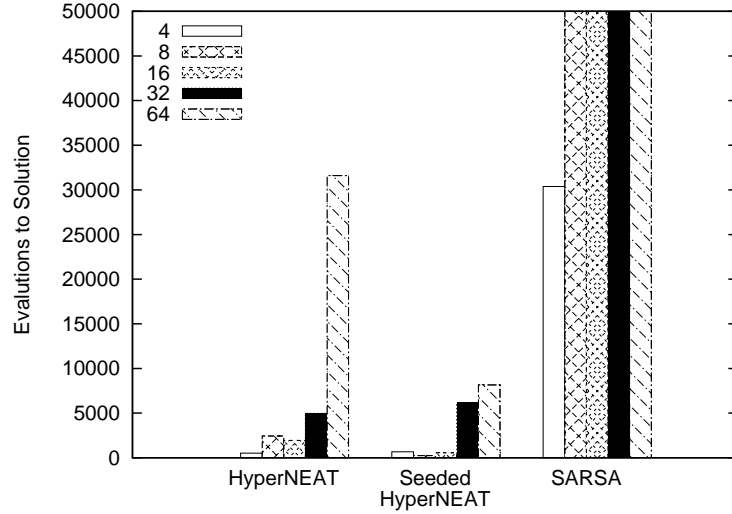


(a) Time to Find Solution

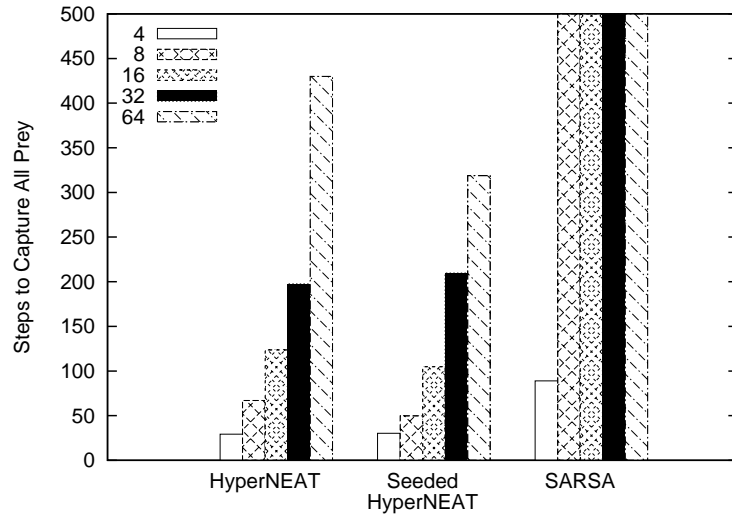


(b) Time to Capture Prey

Figure 8.2: Line Pre-Training Scaling. The performance of each method for pre-training scaling on the line formation is shown. In almost all cases multiagent HyperNEAT teams are able to find better solutions more quickly than SARSA( $\lambda$ ). After 16 agents, SARSA( $\lambda$ ) can no longer solve the task, while HyperNEAT is still able to solve it up to 256 agents. Results are averaged over ten runs.



(a) Time to Find Solution



(b) Time to Capture Prey

Figure 8.3: L Pre-Training Scaling. The L formation was more difficult than the line due to the exploration necessary to solve the problem. However, multiagent HyperNEAT was able to find solutions consistently for all tested sizes. SARSA( $\lambda$ ) could only find solutions at the smallest size of four, and seeded SARSA( $\lambda$ ) was unable to find any solutions. Note that because the distance between the furthest prey and the predators doubles each time the size increases, it is also reasonable for the time to capture the prey to double as well. Results again are averaged over ten runs.

implemented properly; the competitive results at the smaller scales provide evidence that the disparity at higher scales is not only implementation-dependent.

For the team sizes that both methods could solve, multiagent HyperNEAT was only significantly faster in terms of capture speed at eight and 16 on the line formation and four for the L ( $p < 0.01$ ), implying that at smaller sizes both methods are able to refine existing solutions once they are found, but actually finding an initial solution with separately-represented agents is a more difficult problem. Furthermore, for a two-predator team, the smallest size, SARSA( $\lambda$ ) finds solutions significantly faster ( $p < 0.01$ ) than multiagent HyperNEAT, implying that reinforcement learning may be more efficient for very small teams.

Seeding generally improved multiagent HyperNEAT’s performance, although there are some instances where both seeded and unseeded teams were able to find an optimal solution or where seeding hurt performance. With SARSA( $\lambda$ ), seeding was only beneficial at size 16 on the line, indicating that just having a good seed policy may not be enough; the way in which these policies are manipulated during learning is also critical.

Post-training scaling is more challenging: More agents are added to the team without further training, which means the policies of the new agents must somehow be intelligently derived from the old. Because HyperNEAT encodes the team as a pattern, it can create the policies of the new agents by querying the CPPN at an intermediate location, thereby creating an interpolation of the existing policies. Multiagent SARSA( $\lambda$ ) has no such mechanism to automatically generate new policies. In this sense it is difficult to compare them when SARSA( $\lambda$ ) lacks an analogous capability. However, it is still an important question



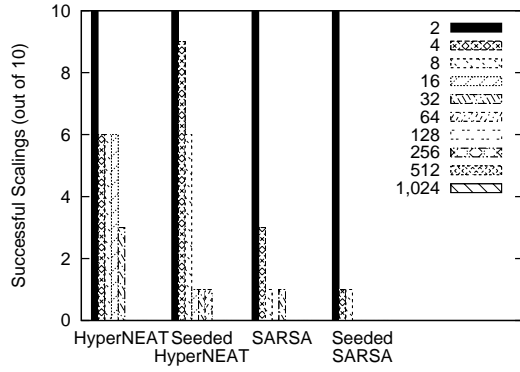
whether interpolating new policies gains any real advantage over simply duplicating the range of existing policies to make a larger team. Therefore, to validate the contribution of role interpolation, scaled SARSA( $\lambda$ ) teams contain multiple duplicates of the agents in the unscaled version, as described previously. The post-training results were obtained by testing all saved champions (i.e. teams that scored better than all previous teams during the run) on all sizes to determine the best scalers for each run. This method of testing generalization follows Gruau et al. [GWP96] and is designed to compare the best overall individuals. In addition to scaling up, scaling down is also tested. Because the number of opportunities to scale up or down depends on the training size being tested, teams are tested and evaluated separately on scaling up and down, but they are displayed together.

Figures 8.4 and 8.5 show post-scaling results on the line formation and figure 8.6 shows results on the L. Each graph shows the number of runs where the best scaler from a run can scale to that size out of ten runs starting from teams trained at different sizes. If a method was not able to find a solution at a particular pre-trained team size then that size is not included as a starting point for post-scaling training. For the line, SARSA( $\lambda$ ) was only able to scale from the smallest two-agent size, and only some runs were able to find scalable solutions. In contrast, multiagent HyperNEAT found scalable solutions from all starting sizes (except in the case of 64 on L, which is the largest size L possible within the time limit). For small team sizes, multiagent HyperNEAT was consistently able to find solutions that can scale at least up to the next team size, although in most cases teams scaled several sizes up and down. The results were similar on the L, with the exception of 64, because there

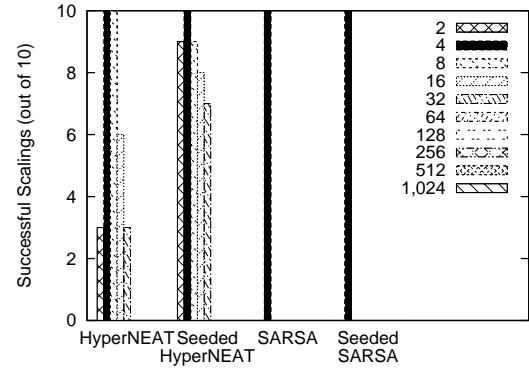
is simply not enough time to capture all the prey within the time limit at 128 agents on the L. Scaling up above 128 agents on the line proved to be difficult for multiagent HyperNEAT, which typically only found solutions that could scale to such sizes one to three times out of ten. However, *seeded* multiagent HyperNEAT was able to find solutions that scaled up to 1,024 agents, a team size that no method could solve when trained to solve it. Thus policy interpolation produces a significant new potential to scale already-trained teams.

### ***8.2.1 Post-training Scaling with Further Learning***

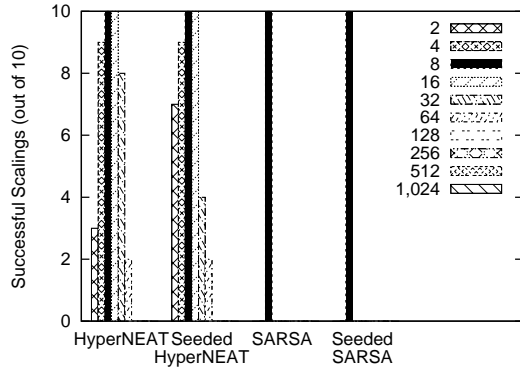
While it is interesting from a research perspective that multiagent HyperNEAT can scale to different team sizes without further learning, as a practical matter, there is no reason that the teams cannot continue learning at the new size to further optimize or correct minor imperfections in the scaled policy. In fact, while many policies generated by multiagent HyperNEAT are able to scale to different team sizes, those that do not scale perfectly still generally display an appropriate (though not perfect) strategy based on policy geometry, indicating that the scaled policy is close to a correct solution but may exhibit some artifacts in the policy geometry that were not sampled at the training size. Previous HyperNEAT research (Chapter 3) showed that similar artifacts are present when scaling the sensors and effectors of a single agent, but that such artifacts are easily smoothed by additionally learning. To test that scaled teams trained with multiagent HyperNEAT can be further trained to



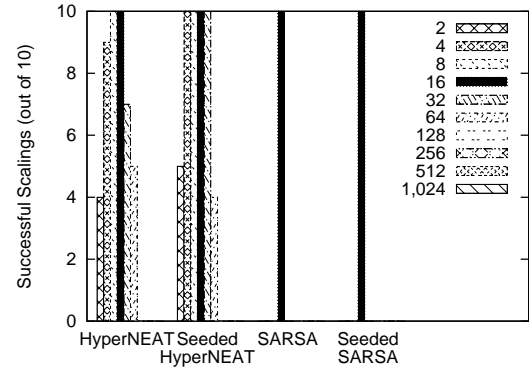
(a) 2 Agent Scaling



(b) 4 Agent Scaling



(c) 8 Agent Scaling



(d) 16 Agent Scaling

Figure 8.4: Line Post-Training Scaling 2-16 (higher is better). The number of successful scalings (both up and down) out of ten runs for different team sizes is shown. For each graph the training size is shaded black. While HyperNEAT is able to find scalable solutions at all sizes without further learning, SARSA( $\lambda$ ) can only scale from two-agent solutions (up to at most 16).

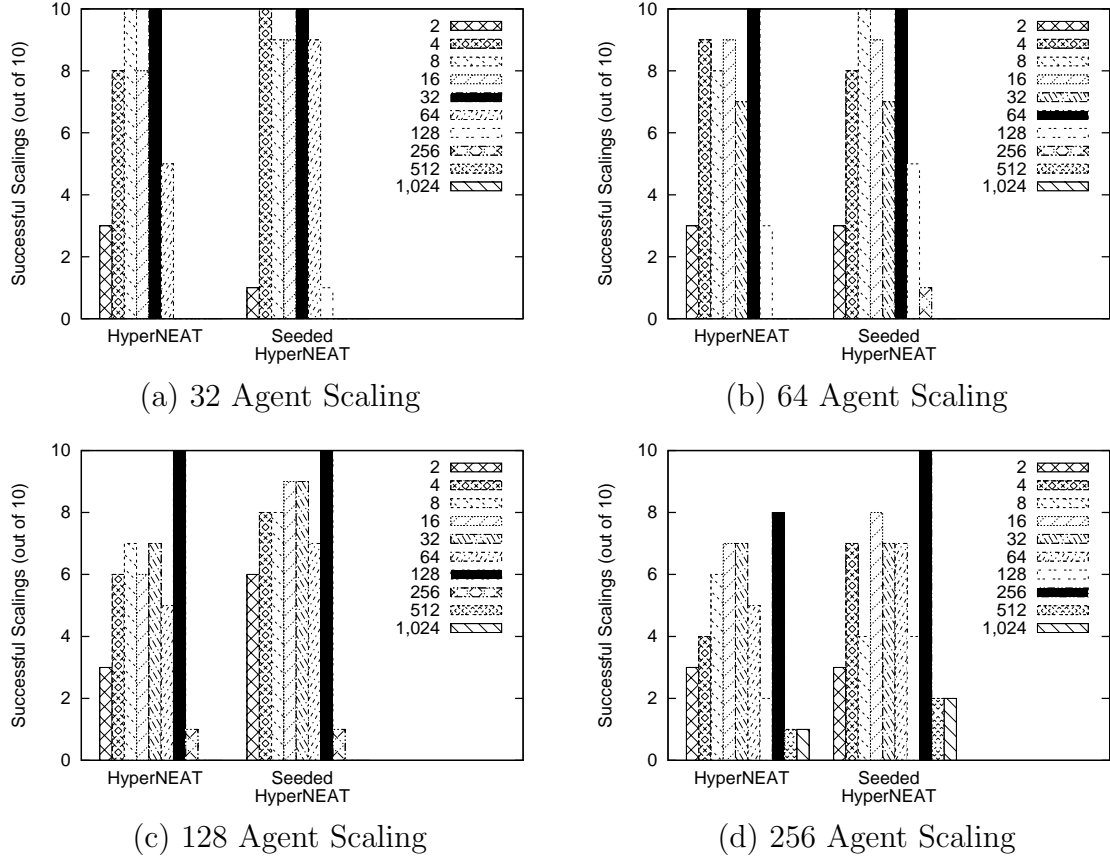
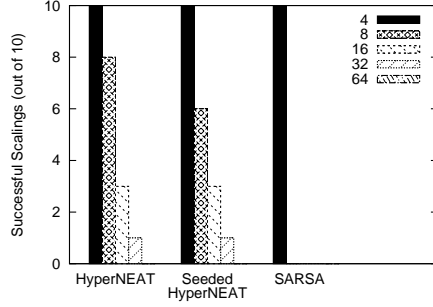
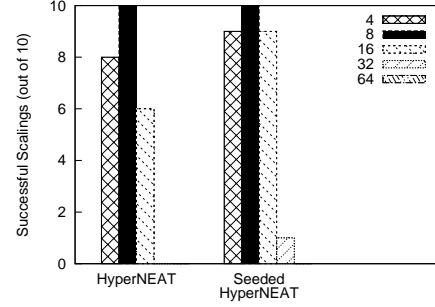


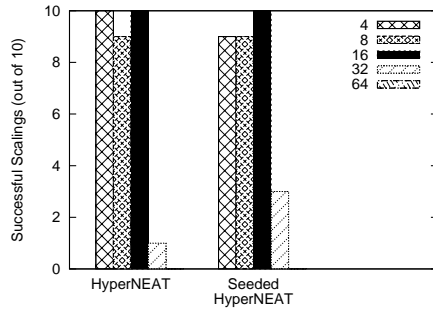
Figure 8.5: Line Post-Training Scaling 32-256 (higher is better). Again, the number of successful scalings (both up and down) out of ten runs is shown, but for the larger team sizes that SARSA( $\lambda$ ) could not solve during training. For each graph the training size is shaded black. Scaling up at larger sizes becomes more difficult because of the different strategies required for different sizes and the sheer magnitude of the increase in agents (e.g. 256 to 512 versus four to eight). Nevertheless, multiagent HyperNEAT is able to find solutions that scale, even up to 1,024 agents, a size that could not be learned when specifically trained to do so.



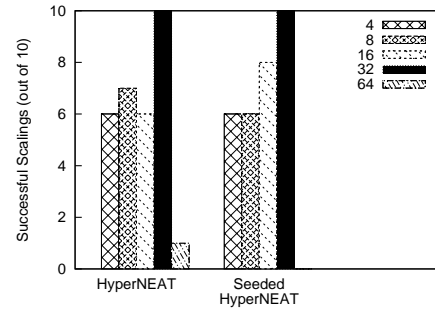
(a) 4 Agent Scaling



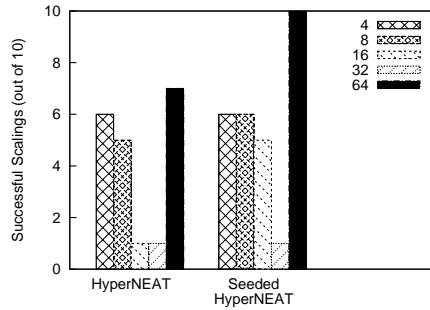
(b) 8 Agent Scaling



(c) 16 Agent Scaling



(d) 32 Agent Scaling



(e) 64 Agent Scaling

Figure 8.6: L Post-Training Scaling (higher is better). Successful scalings (out of 10) are shown on these graphs for different team sizes. The training size that is scaled from is shown in black. HyperNEAT was generally able to find scalable solutions for all sizes, while SARSA( $\lambda$ ) could not scale at all.

correct imperfections, the three best scaling teams at size 64 are further trained at size 256 (to which no unseeded team trained on 64 agents could scale without further learning) on the line formation. Although SARSA( $\lambda$ ) was not able to train a team of 64 agents, to facilitate some comparison, and to test whether this capability is unique to HyperNEAT, teams trained by SARSA( $\lambda$ ) with four agents are scaled to eight agents and further trained.

Multiagent HyperNEAT was able to find solutions that solved the new size in 1,220 evaluations on average, which is significantly faster than finding a solution from scratch on 256 agents (which takes on average 24,650 evaluations). In contrast, the number of evaluations for SARSA( $\lambda$ ) to find a solution at size eight after scaling (average 4,764 evaluations) did not differ significantly from the number required to find a solution for eight agents from scratch (which takes on average 3,804 evaluations). This measure of performance is also similar to the jump-start measure for transfer learning [DAR05]; however, the main concern is how many evaluations it takes to reach a certain performance level (i.e. capturing all the prey) rather than measuring an increase in performance over some number of generations. Thus while it is possible for multiagent HyperNEAT to find solutions that scale without further learning, even those that do not scale perfectly still retain the information necessary to solve the problem at different scales; it simply must be tweaked to accommodate the new size. Additionally, the fact that SARSA( $\lambda$ ) does not benefit from this bootstrapping approach implies that policy geometry plays a critical role in further training after scaling.

### 8.2.2 *Typical Behaviors*

This section describes the typical behaviors produced by both learning methods. Videos of some of these behaviors are available at <http://eplex.cs.ucf.edu/comparison.html>.

In the line, on the smallest team sizes (two, four, and eight), the typical strategies employed by multiagent HyperNEAT and SARSA( $\lambda$ ) do not differ greatly. The first solution discovered at these sizes involves one or more agents going behind the group of prey and pushing them towards the remaining agents. On sizes four and eight, teams then transition into some or all of the predators surrounding and capturing the prey agents. At size 16, SARSA( $\lambda$ ) continues to employ the same strategies (figure 8.7a, b, and c), but because the prey line is longer and there are more agents to coordinate it takes much longer to find a solution and for that solution to capture the prey. In contrast, while multiagent HyperNEAT also first finds such solutions, it quickly discovers different strategies that divide the prey into two groups that the predators independently surround (figure 8.7d, e, and f), exploiting the symmetry of the team. Beyond size 16, SARSA( $\lambda$ ) stops solving the problem because a more coordinated approach is needed.

The first solutions multiagent HyperNEAT finds at sizes 32 and 64 tend to be the same strategies as at 16, but at these sizes they are suboptimal and act as stepping stones to strategies that divide the prey into multiple groups that are captured independently (figure 8.8a, b, and c). Multiagent HyperNEAT is able to find such strategies by repeating coordinate frames within the policy geometry.

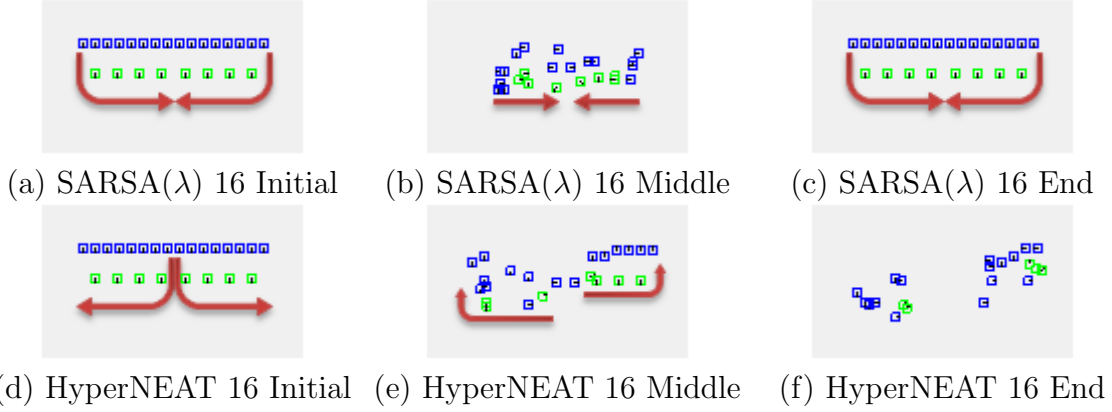
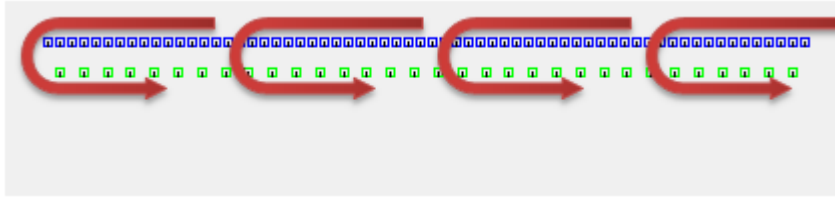


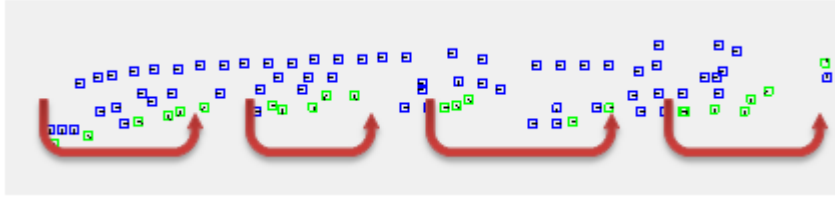
Figure 8.7: Typical Strategies for 16 Predators. At size 16 predators (blue squares at the top of the pictures) SARSA( $\lambda$ )-trained teams typically try to surround the prey (green squares closer to the bottom) by employing the predators on the edges to move to behind the prey (a) and push them towards the center (b) where they can be captured (c). Multiagent HyperNEAT teams also learn this strategy, but eventually develop a more complex version wherein the predators divide the prey into two groups (d) and then surround (e) and capture them independently (f). The multiagent HyperNEAT result is more efficient because more prey are captured in parallel, and it is more scalable because eventually the size of the prey lines becomes too large to traverse within the time limit.

Finally, at sizes 128 and 256 simple strategies are no longer viable due to the length of the line of prey. Thus the first strategies that are found are those that split the prey into multiple groups. Some runs at these large sizes were also able to find an alternative strategy wherein every other predator does nothing and the remaining predators move forward, causing the prey to bounce between each pair, until each prey runs into one of the predators while trying to avoid the other (figure 8.9a, b, and c). Such a strategy is simple and effective, but would be very difficult to learn if all agents were represented independently. Seeded and unseeded versions of the same method did not cause a significant qualitative difference in behavior; better behaviors were just found faster.

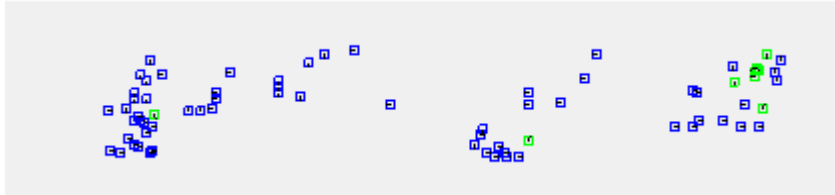




(a) Multiagent HyperNEAT 64 Initial



(b) Multiagent HyperNEAT 64 Middle



(c) Multiagent HyperNEAT 64 End

Figure 8.8: Multiagent HyperNEAT Strategies for Large Teams. For larger teams such as the team of 64 predators in this figure, multiagent HyperNEAT typically continues the strategy of dividing the prey into groups (a), surrounding (b), and capturing them (c). The difference from smaller sizes is that they are divided into more and more groups so that larger numbers of prey can still be efficiently captured by the predators.

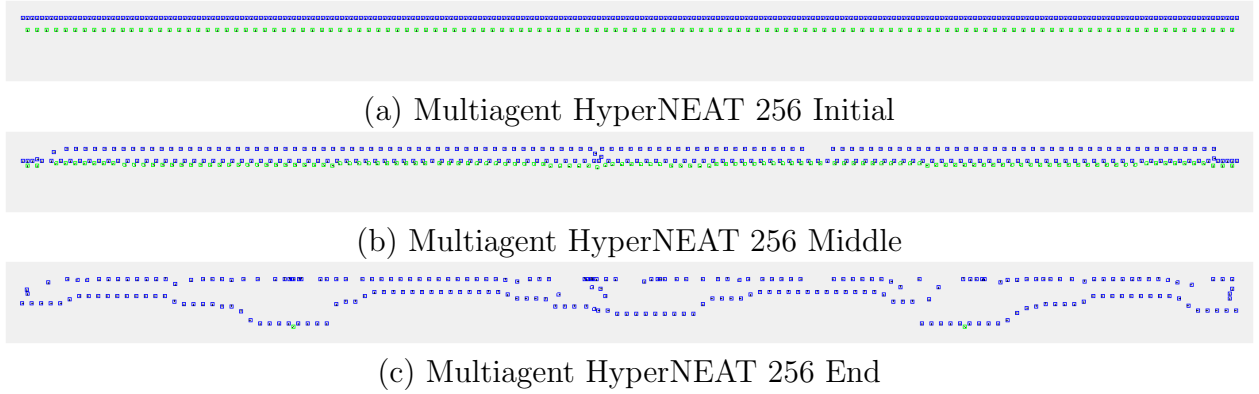


Figure 8.9: Multiagent HyperNEAT 256 Agent Behaviors. At the very largest trained size of 256, multiagent HyperNEAT sometimes found a strategy wherein every other predator does nothing and the remaining predators move forward (a). This strategy causes the prey to bounce between the charging predators (b) until they are all eventually captured (c). Such a tactic is extremely efficient at this problem size and requires strict cooperation of almost every agent to be successful.

On the L, at size four, both methods used the same strategy of surrounding the prey to capture them. Because there are so few prey, simple strategies remain effective. However, at size eight, SARSA( $\lambda$ ) can no longer solve the problem, but multiagent HyperNEAT teams learn a strategy to deal with the vertical portion of the L similar to that of the large teams on the line: The first and third agent move forward and capture the entire line by bouncing the agents between them while the horizontal portion of the L is captured by the remaining predators through a surround strategy. The remaining sizes (16, 32, and 64) capture the vertical part of the line by employing a group of agents to go down one side of the prey, pushing them slightly to the right, while another group charge straight at them. The combination of these two actions forces the prey into a compact ball that is pushed downward by the charging prey. When the first group of prey reach the end of the vertical portion of the L, they turn right forming a barrier between the charging predators and the ball of prey, and

when the two meet, the entire ball is captured. The horizontal portion is again captured by a simple surrounding strategy. There is little incentive for the predators to improve upon the horizontal capture speed at these sizes because the time taken to capture the vertical portion of the prey dominates the problem. Nevertheless, the results on the L formation show that multiagent HyperNEAT could discover effective asymmetric strategies for capturing prey.

The few SARSA( $\lambda$ ) solutions that could scale up post-training did not typically resemble the solutions they scaled from, although the only scalable solutions came from two-predator, one-prey experiments where the overall policy is difficult to discern. In contrast, most scaled multiagent HyperNEAT solutions did resemble the solutions from which they were scaled, but sometimes contain some inefficiencies such as a predator pushing the prey in the wrong direction initially. A key factor in determining whether a team will scale post-training is whether the team employs a strategy that is effective at a higher level. For example, if a team trained at size 16 has the strategy of positioning a single agent behind all the prey and pushing each of them towards the predators, it has less of a chance of transferring to scaled teams. However if the team instead learned how to divide the prey and capture them as groups at size 16, its chances of scaling are much greater.

### 8.3 Comparison Discussion

The result that heterogeneous teams trained with multiagent HyperNEAT significantly outperform teams trained with SARSA( $\lambda$ ) in both training and scaling demonstrates the importance of exploiting team geometry in multiagent learning. Whereas teams trained with SARSA( $\lambda$ ) were unable to solve problems with over 16 agents, multiagent HyperNEAT teams solved up to 256 agents with pre-training. In this way, the ability to encode patterns of behavior across a team is critical to success in multiagent learning and thereby addresses a major challenge in the field. Multiagent HyperNEAT allows team behavior to be represented as variation on a theme encoded in a *single genome*, which means that key skills need not be rediscovered for separate agents, overcoming the problem of reinvention. Furthermore, because multiagent policies are represented by a CPPN, they are assigned to separate agents as a function of their relative geometry, while simultaneously exploiting the agents' internal geometries.

The evaluations taken by multiagent HyperNEAT to find a solution increases greatly at 128 agents. This change reflects a fundamental discontinuity in the policies required to solve the problem at smaller and larger sizes. At smaller sizes, a common early solution is for a small subset of agents to capture all the prey while the others are not involved. While such a solution is suboptimal, it is a solution nonetheless and is a stepping stone to more efficient solutions that make use of the whole team. However, if learning is first to succeed only with a subset of the team, a subset of predators must be able to visit each of the

prey. Yet when there are 128 predators, the line of prey becomes longer than the distance a single predator can traverse in the time limit. Thus, at large sizes, such suboptimal policies represent local optima that cannot lead to efficient solutions and the only viable strategies are the more complex coordinated ones that employ more predators and take longer to find. While HyperNEAT can find these with effort, the required coordination is too much for SARSA( $\lambda$ ).

Multiagent HyperNEAT was able to create teams of agents that could perform well at larger sizes than the training size without further training. Such scaling is prohibitive for SARSA( $\lambda$ ), which could only scale from very small teams. This limitation exhibited by SARSA( $\lambda$ ) in this paper is consistent with past attempts at scaling with MARL, which also only achieved scaling at small sizes (e.g. between four and eight) in a constrained version of this domain [PT09]. Such techniques do not implement interpolated scaling, yet multiagent HyperNEAT accomplishes it by representing the teams as patterns rather than as individual agents. Even though these patterns are sparsely sampled during training (i.e. with orders of magnitude fewer points) the scaled teams exhibit smooth interpolations between agent roles. A particularly interesting result is that while multiagent HyperNEAT could not solve the problem with 1,024 agents when starting from scratch, a team trained with 256 agents could solve the problem when scaled up. This result implies that scaling up may offer more than just the benefit of saving computational time, but also may allow multiagent HyperNEAT to solve problems that are prohibitively deceptive or complex. The impact of deceptive agent

interactions for large teams may diminish at smaller team sizes where such interactions are not as devastating.

However, it is true that multiagent HyperNEAT could not always scale up perfectly post-training. As described above, there are some cases where the policies required to solve a larger size are fundamentally different than those at smaller sizes. This observation separates these results from the previous scaling attempts in Chapter 7 in which the size of the problem (i.e. the number of prey), and therefore the strategy required to solve the problem, remained constant even as team size increased. However, even when the number of prey increases, the team at the smaller size may still encode fundamental regularities that are important to the problem regardless of scale, such as symmetry or the ability to flank a prey. That way, this approach to scaling can still be a good starting heuristic for additional training. Indeed, even when HyperNEAT’s scaling was not perfect, some teams could nevertheless rapidly adapt to a new size when explicitly trained further for it. Thus, while the policy geometry discovered at smaller team sizes may overspecialize or may contain artifacts that were not sampled at the training size, the general patterns remain useful at different team sizes and can be quickly adapted for such different sizes. SARSA( $\lambda$ ) was not able to benefit from this capability because even though it can continue training at new sizes as well, it is blind to the geometry of the team and therefore cannot intelligently extrapolate the behavior of the team as a whole.

In addition to scaling up, teams trained with multiagent HyperNEAT were able to scale down, which could allow recalibrating a team after some agents have been damaged. Scaling

down also exemplifies the need for policy geometry because the regularities discovered at larger team sizes can be maintained at smaller sizes. However, scaling down did not always work because the algorithm may have converged onto regularities that are only useful at larger team sizes, or may have relied on a specific agent position that no longer exists. Nevertheless, the ability of a team to dynamically change size, up or down, without further learning is a beneficial feature imparted by multiagent HyperNEAT.

Indeed, an unfortunate consequence of comparison-driven experiments is often an unwarranted emphasis on determining the *better* method. While multiagent HyperNEAT indeed exhibits greater scalability both during and after training, of course it was *designed* from the ground up (i.e. through the scalability of indirect encoding in HyperNEAT) to be able to scale. On the other hand, SARSA( $\lambda$ ) was not designed with this aim in mind, and thus does not acquire it in the multiagent case. At the same time, as an evolutionary approach, HyperNEAT does not offer the online learning capability inherent in value-function-based reinforcement learning such as SARSA( $\lambda$ ). Thus rather than a critique of SARSA( $\lambda$ ), this comparison is better interpreted as a validation that multiagent HyperNEAT brings something new to the table through the idea of policy geometry. In this more cooperative spirit, perhaps a more enlightened approach to interpreting comparisons is to consider that the fruits of divergent research areas may sometimes be complementary, and thus present opportunities for cross-fertilization. The emphasis on scalability and large size that has been a focus of research in indirect encoding within evolutionary computation for many years has of yet attracted little attention in the reinforcement learning community in general. Per-

haps this study can help to begin to bridge this gap by showing what we all might gain by beginning to communicate despite our foundational differences.

For example, an intriguing possibility is that indirect encodings such as CPPNs can be combined somehow with traditional approaches. For example, perhaps MARL can work at the level of a team instead of on individual agents and reinforcement signals can modify the weights of the CPPN. Yet this prospect can only be realized if a method is devised to propagate the reinforcement error signal through the level of indirection between the substrate ANN and the CPPN that encodes it. At present, no such *indirect error propagation* technique exists, though the possibility is open. Thus, at present, multiagent HyperNEAT is unique in its capability to scale teams through policy geometry interpolation.

The next chapter demonstrates multiagent HyperNEAT in a real-world task with real robots and introduces an extension to the idea of policy geometry that allows each agent to encode multiple policies among which they can choose based on their current state.



## CHAPTER 9

### REAL-WORLD MULTI-TASK LEARNING

While multiagent HyperNEAT has proven successful in many simulated domains, both in this dissertation and in publications by others [KGM10], it has not yet been demonstrated in the real world. In this chapter, teams trained in simulation with multiagent HyperNEAT are transferred to real Khepera III robots to perform a patrol and return task. This domain requires a team of agents to cooperatively cover an environment as in other multiagent patrol tasks [ARS04, SRC04, MBC10], but an important difference in this version of the task is that each agent receives a signal that, when activated, tells them to return home. Thus each agent must perform two separate, yet related tasks. This factor poses a problem for cooperative multiagent learning because of task interdependencies; if a single robot fails to perform as expected the entire team can fail.

There are many approaches to solving problems in which agents must learn to perform multiple tasks. One important strategy is decomposing the main task into hierarchies of subtasks [MMG01]. In this approach agents can focus on these specific subtasks and complete them according to the hierarchy. However, the tasks must be decomposed by the experimenter. Another method is to learn modular controllers [Nol97] wherein different parts of the controller (e.g. different sets of outputs) are active depending on the state of the robot.

Evolving *adaptive* artificial neural networks (ANNs) is another significant technique [FU00], wherein local learning rules facilitate the policy transition from one task to the other. Despite the abundance of methods, task switching is still a difficult problem, especially for cooperative multiagent learning.

To address this challenge, this chapter introduces an extension to multiagent HyperNEAT and the standard policy geometry concept called *situational policy geometry* that generates *multiple policies* for the same agent, among which it can switch depending on its current state. That way, individual agents can learn how to perform multiple *tasks* by learning how they relate to each other, which is similar to the way multiagent HyperNEAT learns the policies of multiple *agents*. The next section explains the mechanisms by which multiagent HyperNEAT can exploit situational policy geometry.

## 9.1 Situational Policy Geometry

Situational policy geometry allows agents to learn to switch to different policies depending on their current state. That way, not only can multiagent HyperNEAT exploit similarities among tasks, but the solutions for individual subtasks are likely to be simpler (and thus more easily discovered) than a single policy that must solve all tasks.

To exploit situational policy geometry the CPPN must be made aware of it. Recall that a team is a stack of single-agent substrates (figure 9.1a shows one such substrate). In a

multitask problem, this one network would need to be able to perform each task. To let the network in figure 9.1a know which task is current, it has a special input  $S$ , which is a task signal. However this network is only one member of a team. Following the multiagent HyperNEAT approach inputs are added to the CPPN so that it can represent the standard policy geometry that represents dimensions of the team (figure 9.1b). In a similar way, to implement situational policy geometry, inputs are added to the *CPPN* so that it can represent the dimensions of the *tasks* (figure 9.1c). For example, in the experiment in this chapter, a single new dimension  $S$  is added, which is either 1 or -1, depending on whether the robot must come home or not. Because the task signal is now a part of the CPPN (figure 9.1c), the substrate controller for an individual robot no longer needs the signal input (figure 9.1d). Also, because the CPPN now has an extra dimension, there are now two stacks of controllers (one for each value of the signal) instead of one (figure 9.1e). In effect, each agent now has two brains that it can switch between depending on their current task.

## 9.2 Patrol and Return Experiment

To demonstrate that multiagent HyperNEAT can produce teams that are robust enough to function in the real world and to explore the capabilities of situational policy geometry, teams are evolved to solve a patrol and return task in which robots must spread out and observe an environment and then return home when signaled to do so. Patrolling tasks are common

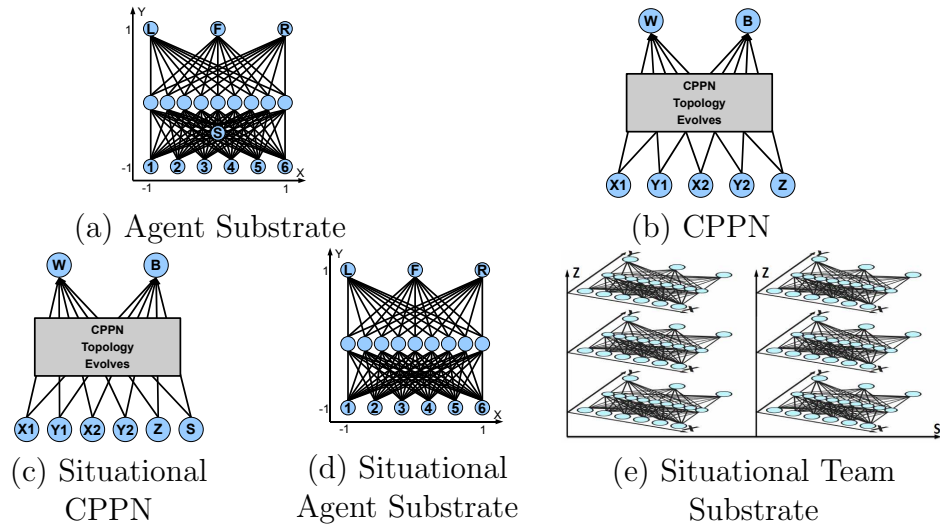


Figure 9.1: Multiagent HyperNEAT with Situational Policy Geometry. In standard multiagent HyperNEAT, a substrate (a) is changed into a multiagent substrate by adding an additional input(s) to the CPPN (b) that represents the policy geometry of the team. Situational policy geometry is handled similarly: The CPPN (c) has an additional input  $S$  that describes the location in the situational policy geometry for a given agent’s controller (d), which is formalized in the new  $S$ -axis in the situational team substrate (e). Thus the network stack to the left along  $S$  is team policies for one situation while that on the right is policies for a different situation.

in multiagent learning [ARS04, SRC04] because they require agents to cooperate to ensure that they do not collide with each other and to achieve uniform coverage of the area. The task in this experiment is made more complex by the fact that the robots must return home on command, meaning that each agent must effectively learn two roles and remember how to return home. This requirement also makes the task more realistic: If a group of robots was sent to patrol a building, recalling them after a period of time to recharge batteries or when the patrol is over would be preferable to manually collecting the robots.

Unlike other approaches to multiagent patrolling [MBC10, ARS04] the robots in this task cannot communicate with each other. This limitation means that the agents must learn *a priori* roles to maximize coverage and minimize overlaps. By exploiting the policy geometry, multiagent HyperNEAT can accomplish this goal by finding a general patrolling and collision-avoidance policy for all the agents, and at the same time by varying the policy for each agent so that they patrol different areas. Also, the patrolling robots must respond to a “come home” signal, requiring a robust dual policy that keeps them deployed until the signal, at which point they must quickly return home. By exploiting situational policy geometry, the idea is that robots can employ separate yet related policies for these conflicting tasks. Experimental parameters are given in the Appendix.

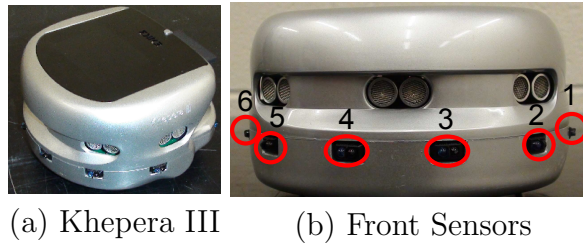


Figure 9.2: Khepera III with Korebot II. The Khepera III mobile robots (a) in these experiments come equipped with a Korebot II extension that runs an embedded Linux operating system and allows the robots to receive broadcast communications over a wireless network. Although the Khepera III has many sensors available, only the front six infrared rangefinders (b) are utilized in these experiments.

### 9.2.1 Robots

The robots used in this experiment are three Khepera IIIs outfitted with KoreBot II extensions (figure 9.2a), which make it possible for the neural network controllers to run on the robots themselves, thereby minimizing command latency and reducing the need for communication with a base station to only general broadcast signals (i.e. start and return). The Khepera III is equipped with both long-range, ultrasonic and short-range infrared rangefinder sensors; however for this task only the front six infrared sensors are utilized (figure 9.2b). The sensors can detect both walls and robots, but *cannot* distinguish between them. The Khepera III can achieve speeds of up to 30cm/sec, but because of the size of the environments and to avoid damage to the robots during testing the motors were run at a reduced speed with an approximate velocity of 6cm/sec. For more information on the Khepera III see <http://www.k-team.com/>.

Each robot is controlled by a separate neural network (either figure 9.1a or d depending on whether or not they are learning situational policy geometry) generated by the same CPPN (figure 9.1b or c). If the robot is using situational policy geometry, it has six input nodes corresponding to the six rangefinder sensors. The robots without situational policy geometry have one additional input (called  $S$  in figure 9.1a) that indicates whether the robot should return home or continue patrolling. The rangefinders on the robot return values between 0 and 4,000; larger numbers represent farther distances. Preliminary experiments indicated that the sensor values beyond four inches are very noisy and that the response curve of the sensors is non-linear. Therefore, before the values are fed into the neural network, the raw sensory input is modified to clip values beyond four inches, to be more linear, and to be scaled between zero and one through the function  $1.43 - \frac{(\log(s)-0.51)}{2.25}$ , where  $s$  is the raw sensor value returned by the robot.

A robot can select from one of three actions: go forward, turn left, or turn right. The action is selected based on the values of the three network outputs in figure 9.1a or d; the output with the highest value is the action for that timestep. The robots are allowed to select actions every 33 milliseconds because that is the update rate of the Khepera III's infrared sensors. Robots also have a collision avoidance policy to reduce the chance of damaging themselves that overrides the robot's forward command: If either of the two leftmost sensors fall below 0.25, the robot turns right; the opposite is true for the two rightmost sensors and if either of the front sensors fall below the threshold, the robot stops.

Training teams through artificial evolution with real robots would be time consuming and could potentially damage the robots. Instead the teams are trained in a custom simulator made in our research group (available at <http://eplex.cs.ucf.edu/software.html>). Only simple two-dimensional kinematics are simulated with an update rate of 33msec. This approach is faster than modeling and simulating three-dimensional motion and/or realistic physics, and was nevertheless found to be just as accurate as transferring from e.g. Webots [Web] in real-world transfer in preliminary experiments. The Khepera IIIs are modeled based on manufacturer specifications and preliminary calibration experiments.

### 9.2.2 *Environments*

Each team is trained in the same environment to encourage robots to adopt specific roles, but in the real world they are also tested on a variant of that environment to ensure generality. The training environment is called *the plus* (figure 9.3a) and is made of an entrance that leads to three branching paths. Branches are approximately 29cm wide and 77.5cm long. To cover this environment, the robots must split up and take separate paths, even though they cannot communicate with each other. Thus the robots must have some *a priori* bias that allows them to cooperate. The testing environment is called *the asymmetric plus* (figure 9.3b) and is similar to the plus, but with several changes. First, the left path is shortened to approximately 48.43cm and the right and center paths are 1.5 times as long as in the regular



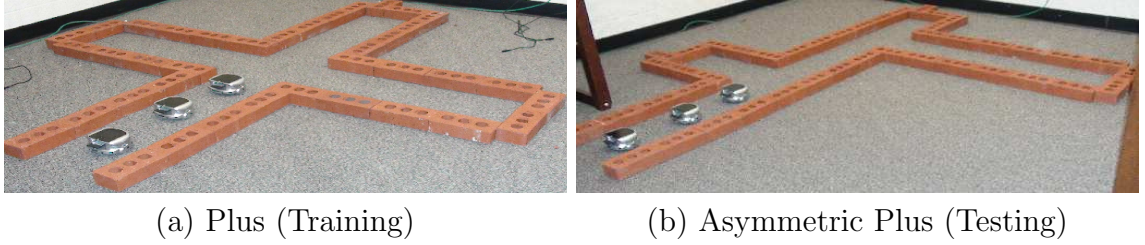


Figure 9.3: Real-World Environment. The real environments with which the robots interact are constructed out of red bricks on a carpet with the same dimensions used in the simulator. The plus (a) is the environment the robots are trained on, and the asymmetric plus (b) tests the generality of the learned policies. In both cases robots are placed 30cm apart in the open branch and then sent a signal to begin patrolling. Individual robots can then be called back by the experimenter by broadcasting a command to them.

plus. Also, the right branch is shifted up by 77.5cm. These changes cause very different sensor activations where robots would typically turn and stop, thereby testing the generality of the learned policies. The environments are designed to capture the general idea of patrolling, while not being too complex to build physically (out of bricks) in the real world.

For the real robots, the environments are constructed out of red  $7\frac{5}{8}\text{in} \times 3\frac{5}{8}\text{in} \times 2\frac{1}{4}\text{in}$  bricks with a carpet base, which are the same dimensions as in the simulator. The three robots are placed in the starting branch of the environment, 30cm apart. They are then simultaneously started and begin patrolling. A good solution is for all agents to reach the end of a different branch and stop. After all agents are stopped, they are called back by activating their “come home” signal in the order that they left. Only one agent is called back at a time to maintain as much coverage as possible. When the agent returns home, its signal is turned off and it is placed back at the home point facing the environment so that it can return to patrolling and the next robot can be called back.

In simulation fitness is assigned to each team based on two criteria: If a robot receives the signal to come home, minimizing distance to home is rewarded, but if it does not yet have the signal, minimizing distance to the end of a hall is rewarded. For every simulated second each robot is given a score of  $\frac{D-d}{D}$ , where  $D$  is the maximum possible distance to either the end of a hall or home, depending on the state of that robot’s signal, and  $d$  is the current distance to that objective. If the robots have not reached the end of the hall when the signal activates, their fitness for returning home is divided by ten, so that solutions that never leave home are discouraged. Similarly, teams in which all agents do not change position or heading after they receive the signal or were still moving forward when they received it have their fitness for patrolling divided by ten to encourage them to respond to the signal. These scores are summed over each robot over each second (out of 45) to give the overall fitness of a team for that trial. Thus the maximum fitness is three times the number of seconds of the trial, although in practice such a fitness is not possible to reach because the robots spend time moving between points. To simplify training, evaluations in simulation are carried out slightly differently than in the real world: Instead of calling the robots one by one, all robots are called simultaneously when half of the evaluation time has passed and robot-to-robot sight and collision are turned off. This method tests the essential requirements of the policies to return home, while speeding up evaluation significantly.

### 9.3 Patrol Results

Fifteen independent runs of multiagent HyperNEAT were conducted with 1,000 generations each in the simulated plus environment with situational policy geometry and with standard policy geometry. In the simulator, a *solution* to an environment is a policy that successfully navigates each of the three robots to the end of a different wing of the map, where they each wait until a signal is sent to them, and then navigate back to the starting location after the signal is received. In each of the fifteen runs with situational policy geometry a solution was evolved in 264.6 generations on average (stdev=246.28). However, with standard policy geometry only three solutions were evolved. This difference is significant ( $p < 0.0001$ ; Fischer’s exact test), highlighting the advantage that can be gained from recognizing and exploiting situational regularities.

Note that evolution did not stop when the first solution was found, so each successful run actually produced a number of viable solutions. To determine which solutions from these successful runs to evaluate in the real world, a generalization test was developed. This test averages the performance of an evolved policy in 25 additional evaluations on the plus environment with varying levels of noise in the robots’ sensors, stochastic robot turning and forward locomotion, and small random perturbations of the initial location and heading of the robots. The idea is that the policies that are more general will perform better in the real world because they will be more robust to the inevitable slight discrepancies between an imperfectly modeled simulated environment and reality.

Confirming this motivation, the five most general solutions from distinct runs with situational policy geometry were all successfully transferred from simulation to the real world, where the solution criteria was even more strict to make sure teams are genuinely robust in real robots without further training: Each robot must go out to its proper position, return home upon receiving the signal, and then return back to its position. In addition, when tested in the real world in the asymmetric plus environment, for which they were *not* trained, these five most general solutions still successfully patrolled and returned, thereby demonstrating that the policies learned by multiagent HyperNEAT with situational policy geometry were not specific to a single map. Videos of such successful transfers from simulation to both the real world plus and asymmetric plus environments are available at <http://eplex.cs.ucf.edu/patrolling.html>. Of the three runs without situational policy geometry that solved the task, two transferred successfully to the real-world symmetric environment, and only one solved the asymmetric environment. Because so few runs could solve the task at all without situational policy geometry even in simulation, the sample size is too small to draw significant conclusions on transferability of such solutions. However, because the chance of even finding a solution is statistically so much smaller without situational policy geometry, in effect if the aim is to find a real-world solution, situational policy geometry provides a significant advantage.

## 9.4 Patrol Discussion

Teams that were trained with situational policy geometry outperformed those trained without it in both simulation and in the real world. One reason for this advantage is that by giving multiagent HyperNEAT situational policy geometry information, it was able to exploit the regularities of the tasks. For example, a major difference between leaving to patrol and coming back is the direction the robot must turn. Agents without situational policy geometry must utilize the value of a specific input to decide how to turn, but those with it utilize a different neural network once the signal fires. A common strategy for situational teams was simply to invert connection weights in the network, allowing them to keep a similar policy, but with an opposite turn bias.

Because they can exploit situational regularities, the policies of the teams with situational policy geometry were simpler than those represented by the teams with standard policy geometry. That is, the agents without situational policy geometry must effectively encode in the same network both how to perform the tasks and how to switch between them. Thus, it is possible for the robot to encounter situations (e.g. those conflated or obscured by noise in robot sensors) that cause it to switch tasks when it is not appropriate. In fact, a frequent failure of these teams in the real world was that they would come back too early or not come back at all. In contrast, the situational teams with their simpler task policies did not exhibit this behavior and were able to transfer consistently to real robots in both the training and testing environments. Thus these experiments verify the utility of breaking up complex tasks

into simpler subtasks as in previous work and offer a new method by which to learn these subtasks that exploits the regularities among them.

Another consequence of this work relates to HyperNEAT and generative and developmental systems as a whole. These systems rely heavily on discovering and exploiting the regularities of a problem such as how the leftmost sensor relates to the left turn effector. However, sometimes there is information that is critical to a problem that does not easily fit into these patterns, such as the “come home” signal: There is no simple geometric relationship between the signal input and the sensors and effectors, so it is unclear where exactly it should be placed on the substrate to best exploit the geometry of the problem. By moving the signal to the CPPN, this information is effectively incorporated into the pattern without disrupting the existing geometry. Thus situational policy geometry opens up a new possibility for indirect encodings wherein information that does not clearly fit with existing patterns can still be elegantly incorporated into the encoding.

A future direction for this work is to investigate domains with larger numbers of tasks or subtasks. Multiagent HyperNEAT should be able to discover the relationships between varying numbers of tasks just as it is able to do so among varying numbers of agents (Chapter 7 and Chapter 8). More tasks can be added by either increasing the sampling rate of a single dimension of situational policy geometry or by introducing new dimensions, depending on the relationships of the tasks. Another intriguing possibility is to allow the agent to decide when to switch between tasks through an output that could either determine when an agent wants to switch tasks, or which task (i.e.  $S$ -coordinate) the agent wants to perform.

In this way, a continuum of tasks could be automatically generated by sampling intermediate  $S$ -coordinates, allowing agents to discover new and interesting ways to divide work and cooperate.

## CHAPTER 10

### DISCUSSION AND FUTURE WORK

Multiagent HyperNEAT exploits geometry to create an efficient multiagent learning algorithm that can scale to large numbers of heterogeneous agents. The first section in this chapter discusses the mechanisms by which multiagent HyperNEAT is informed by geometry and how this ability is beneficial to multiagent learning. The next section explores scalability and the final section looks at potential future directions for this research.

#### 10.1 Policy Geometry and the Continuum of Heterogeneity

HyperNEAT has demonstrated that exploiting geometric information is effective across a variety of domains. This idea is particularly applicable to multiagent teams because they present opportunities to exploit geometry not only within each agent with respect to its sensors and effectors, but also across the team as a whole. This concept of a *policy geometry* is inspired by real teams in which the position of an agent generally reflects its role and behavior. Highlighting the importance of team geometry, in Chapters 6, 7, and 8 multiagent HyperNEAT consistently outperforms methods that are blind to the geometry of their teams.



Such methods that cannot exploit the team geometry must separately learn the policy of each agent, which leads to the *problem of reinvention*. That is, strategies discovered by one agent must be independently discovered by all other agents who require them. The idea of policy geometry circumvents this issue, allowing multiagent HyperNEAT to share discoveries among agents instantly.

While most multiagent learning algorithms strictly separate teams into heterogeneous or homogeneous, multiagent HyperNEAT treats those two categories as only the extremes on a continuum of heterogeneity, allowing the learning algorithm itself to decide the best level of differentiation for a team on given problem. Because multiagent HyperNEAT is an indirect encoding, information that is needed by multiple agents need only be discovered and represented once; thus it can then be repeated or varied for all agents on the team. Such a capability enables compact, efficient encodings of large, heterogeneous multiagent teams. This fact is demonstrated by the results from seeding in all the experiments in this dissertation. That is, when given a strong single policy, the team is initially homogeneous, but multiagent HyperNEAT can rapidly vary this single-agent strategy along the policy geometry to create an effective heterogeneous team.

## 10.2 Scaling

Another key capability of multiagent HyperNEAT is its ability to scale, both in terms of the number of agents trained and by adding or subtracting agents after training has already been completed, without any additional learning. Solving large multiagent problems is challenging for some algorithms because each new agent or role must be learned separately; however multiagent HyperNEAT learns the behaviors of all agents on the team simultaneously in one indirect representation. Thus while adding more agents still makes the problem more complex, the space being searched (i.e. the CPPN) remains the same for two agents or two thousand. Chapter 8 showed that while a competing multiagent learning algorithm, SARSA( $\lambda$ ), could only solve problems of up to sixteen agents, multiagent HyperNEAT was able to find solutions of up to 256 agents when learning from scratch, demonstrating the scalability of multiagent HyperNEAT learning.

Multiagent HyperNEAT is not only scalable in terms of team training sizes; teams were also shown to scale *without further learning* after training. This novel capability is made possible by the way teams are represented. Because teams are encoded as patterns there are effectively an infinite number of agents represented for each team, although only a subset are queried during training. In principle, this property can be exploited when training large teams is necessary, but computing power is limited. That is, simulating very large teams is computationally expensive; in effect whole runs of evolution can be performed at small teams sizes in the time it takes to perform only a few generations of large teams. Thus

teams can be trained at small sizes and scaled up later. Additionally, it may be easier to discover fundamental regularities in the policy geometry such as symmetry when the team is smaller and there is less deception caused by interactions among the agents. For example, while multiagent HyperNEAT was not able to train a team of 1,024 predators to capture 512 prey from scratch (Chapter 8), team sizes that were an order of magnitude smaller were able to solve the problem when scaled up to that size because they were able to recognize the important aspects of policy geometry.

### 10.3 Future Work

One aspect of multiagent learning that has not been explored in this dissertation is communication among agents. Focusing on agents that do not communicate made it possible to focus on the *a priori* effects of policy geometry. In principle, some problems may be impossible, or at least much more difficult to solve, if agents are not allowed to pass messages to each other. However, communication can also be seen as part of an agent’s policy, and an interesting research direction is to incorporate such communication into multiagent HyperNEAT’s current description of policy geometry. In fact, the capability to do so already exists within the algorithm: Currently the weight of a connection is described as  $CPPN(x_1, y_1, x_2, y_2, z)$ , which defines the connections within an agent; however if an additional parameter  $z_2$  is added (giving  $CPPN(x_1, y_1, x_2, y_2, z_1, z_2)$ ), connections *between* agents can be queried. Such con-

nections could be used for communication and could vary with the policy geometry, with the added benefit that the messages would be *in the language of the brain*. That is, communication would be in the form of neural signals that are the same as those within a single neural network, thus eliminating the need for the experimenter to define a separate language and potentially making it easier for the agents to integrate information from each other.

While this dissertation has argued that policy geometry based on an agent’s canonical location is a powerful means of encoding a team, there are situations where a physical location may not be enough information to completely determine an agent’s role. Situational policy geometry in Chapter 9 was one such case, but another idea is that roles can be derived from other properties such as the heterogeneous capabilities of the agents. For example, if two types of UGVs, wheeled and treaded, should cooperate, their differences in the ability to traverse terrain and in speed may be more important than their starting location for determining which agents should do what. Multiagent HyperNEAT has the capability to represent such teams by implementing a *conceptual* policy geometry based on agent capabilities or skills instead of (or in combination with) the physical policy geometry discussed thus far. To do so, new policy geometry axes could be added, such as strength and speed, and agent policies could be queried based on where their traits fall on those axis. Additionally, if agents with different combinations of such capabilities are introduced, their policies could be interpolated based on the existing agents’ policies.

## CHAPTER 11

## CONCLUSION

This dissertation introduced a neuroevolutionary method called HyperNEAT, as well as an extension, multiagent HyperNEAT, a new approach to multiagent learning that encodes teams of agents as patterns based on *policy geometry*. That way it can overcome the *problem of reinvention* because it does not need to discover separate policies for each agent. This technique was tested in several multiagent learning domains (i.e. predator-prey, room-clearing, and patrol), extended to include the ability to scale multiagent teams *without further learning*, and transferred into Khepera III robots to perform a real-world multiagent patrolling task. This chapter summarizes the main contributions of the dissertation.

### 11.1 Contributions

In summary, by showing that multiagent HyperNEAT can scale to very large team sizes with and without further learning and can transfer into real-world teams, this dissertation validates the research hypothesis that a multiagent learning method that exploits policy

geometry and the continuum of heterogeneity can create large, robust, and scalable heterogeneous teams:

1. This dissertation introduced the indirect encoding-based algorithm HyperNEAT, which was co-invented by myself, Jason Gauci, and Kenneth O. Stanley. Despite only being introduced in 2007, HyperNEAT has already proven successful in a large number of domains [GS08, GS07, GS10a, GS10b, DLR10, DS07, DS08, RS10, CBO09, CBP09, CPO09, CBM10, COP08, VS10b, VS10a, SDG09] by researchers both at the University of Central Florida and from outside. HyperNEAT exploits the inherent geometry in problems to influence learning, which paves the way for the major contribution of this dissertation.
2. Multiagent HyperNEAT extends HyperNEAT to allow a single genome to encode multiple, related neural networks. The main idea behind the method was inspired by real teams of agents that tend to display a *policy geometry*, that is, the position of agents relative to others on the team tend to dictate individual agent strategies. Multiagent HyperNEAT exploits this observation by encoding teams as a pattern of related policies based on agent location. In this way, multiagent HyperNEAT was able to create large heterogeneous teams because information critical to all agents is only necessary to discover once and is easily varied among the agents.
3. Multiagent HyperNEAT was extended so that teams trained by it can be scaled up in size by up to several orders of magnitude *without further learning*. This capability

further exploits policy geometry by inferring the roles of agents added to the team from their positions. These new roles are derived from the patterns learned at smaller team sizes, and then the new agents are interpolated by querying intermediate points on the substrate. This novel technique was shown to be effective at scaling teams in two domains (predator-prey and room clearing).

4. Multiagent HyperNEAT was compared to the traditional SARSA( $\lambda$ ) reinforcement learning method in a version of the predator-prey domain. While SARSA( $\lambda$ ) was only able to find solutions for teams of up to 16 agents, multiagent HyperNEAT could effectively train teams up to 256 agents and could scale, without further learning, to teams of 1,024 agents. Additionally, for the team sizes that both methods could solve, multiagent HyperNEAT found solutions in significantly fewer evaluations for all but the smallest team size. These results not only show that multiagent HyperNEAT is a competitive method for multiagent learning, but they also further demonstrate the benefits of exploiting policy geometry.
5. The concept of policy geometry was extended to include *situational policy geometry*, which allows multiagent HyperNEAT to encode multiple policies for each agent on the team that they can switch between depending on their current state. Similarly to how multiagent HyperNEAT exploits how the policies of agents relate to each other with standard policy geometry, it can exploit the relationships of different tasks with situational policy geometry to allow efficient learning of multiple tasks simultaneously. By exploiting situational policy geometry, multiagent HyperNEAT was able to train

teams of agents to patrol and return on command significantly more consistently than methods that encoded both tasks in the same neural network.

6. Finally, teams trained with multiagent HyperNEAT were transferred into real Khepera III robots and performed a real-world, multiagent patrolling task. This experiment showed that multiagent HyperNEAT is a viable platform for practical applications and that the policies generated by it are robust enough to be used in the real world.

Thus this dissertation demonstrates that geometry plays a critical role in many learning problems, but especially so in multiagent learning, where geometry is not only important in individual agent policies, but also for the team as a whole (as demonstrated by policy geometry). These ideas provide a unique perspective on the problem of learning the policies of cooperative heterogeneous teams: Instead of learning individual agent policies, the computer can instead learn how the policies of the agents relate to each other. While such a technique follows naturally from the capabilities of HyperNEAT, other algorithms can also incorporate the ideas of policy geometry and the continuum of heterogeneity, opening up a new research direction in multiagent learning.

## 11.2 Conclusion

Multiagent HyperNEAT is a new, competitive method for training multiagent teams. By exploiting the concepts of policy geometry and the continuum of heterogeneity, it solved



several cooperative multiagent problems with very large heterogeneous teams and was able to scale these teams without further learning, unlike SARSA( $\lambda$ ). Teams trained by multiagent HyperNEAT were also able to transfer to real Khepera III robots and solve a real-world patrolling task.

## **APPENDIX: PARAMETERS**

This appendix contains the parameter settings for the experiments described in this dissertation. Because HyperNEAT differs from original NEAT only in its set of activation functions, it uses the same parameters [SM02b]. Experiments were run using a modified version of the public domain SharpNEAT package [Gre06]. They were found to be robust to moderate variation through preliminary experimentation. The same parameters are used in all experiments, with a few minor exceptions. They are shown in Table A.1.

This section also provides a detailed explanation of each parameter.

1. **Population Size:** The number of networks evaluated every generation.
2.  $c_1$ : A parameter from the original NEAT that defines the weight of excess genes when computing compatibility between networks.
3.  $c_2$ : A parameter from the original NEAT that defines the weight of disjoint genes when computing compatibility between networks.
4.  $c_3$ : A parameter from the original NEAT that defines the weight of connection strength differences when computing compatibility between networks.
5.  $c_t$ : A parameter from the original NEAT that defines the compatibility threshold that determines whether networks are in the same species. In SharpNEAT and Hyper-SharpNEAT, this value is variable and changes to accommodate the desired number of species.

6. **Add Link Probability:** The probability of adding a new connection to a network (i.e. to the CPPN).
7. **Add Node Probability:** The probability of adding a new node to a network (i.e. to the CPPN).
8. **Target # of Species:** The desired number of species in the population. If there are more or less species,  $c_t$  will be adjusted down or up respectively in an effort to maintain the target number of species.
9. **Elitism:** The top percentage of each species that will be copied into the next generation unchanged.
10. **Expressions Threshold:** The value for which a CPPN's output must exceed for a connection to be expressed.
11. **CPPN Weight Range:** The range of weights that the CPPN can assign to the connections in the substrate. The output of the CPPN will be scaled between these values.
12. **Function Set:** The set of functions that the CPPN can choose from when adding a node.

Table A.1: Parameter Settings in Experiments

<b>Parameter</b>	<b>Food Gathering</b>	<b>Predator- Prey (All)</b>	<b>Room Clearing</b>	<b>Patrol and Return</b>
Population Size	150	150	150	500
$c_1$	2.0	2.0	2.0	2.0
$c_2$	2.0	2.0	2.0	2.0
$c_3$	0.2	0.2	0.2	0.2
$c_t$	variable	variable	variable	variable
Add Link Probability	0.03	0.03	0.03	0.03
Add Node Probability	0.01	0.01	0.01	0.01
Target # Species	10 to 15	10 to 15	10 to 15	10 to 15
Elitism	20%	20%	20%	20%
Expression Threshold	0.9	0.2	0.2	0.2
CPPN Weight Range	-3 to 3	-3 to 3	-3 to 3	-3 to 3
Function Set	Gaussian, Sigmoid, Sine, Absolute Value	Gaussian, Sigmoid, Sine, Absolute Value	Gaussian, Sigmoid, Sine, Absolute Value	Gaussian, Sigmoid, Sine, Absolute Value

## LIST OF REFERENCES

- [Aal09] Aaltonen et al. (over 100 authors). “Measurement of the Top Quark Mass with Dilepton Events Selected using Neuroevolution at CDF.” *Physical Review Letters*, **102**(15):2001, 2009.
- [Alt94] Lee Altenberg. “Evolving Better Representations through Selective Genome Growth.” In *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 182–187, Piscataway, NJ, 1994. IEEE Press.
- [Ang95] P. J. Angeline. “Morphogenic Evolutionary Computations: Introduction, Issues and Examples.” In John Robert McDonnell, Robert G. Reynolds, and David B. Fogel, editors, *Evolutionary Programming IV: The Fourth Annual Conference on Evolutionary Programming*, pp. 387–401. MIT Press, 1995.
- [ARS04] Alessandro Almeida, Geber Ramalho, Hugo Santana, Patracia Tedesco, Talita Menezes, Vincent Corruble, and Yann Chevaleyre. “Recent Advances on Multi-agent Patrolling.” In Ana L. C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pp. 126–138. Springer Berlin / Heidelberg, 2004.

- [ASP93] Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. “An Evolutionary Algorithm That Constructs Recurrent Neural Networks.” *IEEE Transactions on Neural Networks*, **5**:54–65, 1993.  
*[kstanley: Evolutionary algorithm with only mutation for TWGANN.]*
- [BBS08] L. Busoniu, R. Babuška, and B. De Schutter. “A comprehensive survey of multi-agent reinforcement learning.” *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, **38**(2):156–172, March 2008.
- [BH97] L. Bull and O. Holland. “Evolutionary computing in multiagent environments: Eusociality.” In *Proc. Ann. Conf. on Genetic Programming. Morgan Kaufmann*, 1997.
- [BK93] Richard K. Belew and Thomas E. Kammeyer. “Evolving Aesthetic Sorting Networks Using Developmental Grammars.” In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. San Francisco: Kaufmann, 1993.
- [BK99] Petet J. Bentley and S. Kumar. “The Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pp. 35–43, San Francisco, 1999. Kaufmann.

- [BKS09] Z. Buk, J. Koutník, and M. Šnorek. “NEAT in HyperNEAT substituted with genetic programming.” In *Proceedings of the International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2009)*, 2009.
- [BM03] Bobby D. Bryant and Risto Miikkulainen. “Neuroevolution for Adaptive Teams.” In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, volume 3, pp. 2194–2201, Piscataway, NJ, 2003. IEEE.
- [Bon00] J.C. Bongard. “Reducing Collective Behavioural Complexity through Heterogeneity.” *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*, 2000.
- [Bon02] Josh C. Bongard. “Evolving Modular Genetic Regulatory Networks.” In *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.
- [BSB05] L. Busoniu, B. De Schutter, and R. Babuska. “Learning and Coordination in Dynamic Multiagent Systems.” Technical Report 05-019, Delft University of Technology, 2005.
- [BTB07] G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, and S. Nolfi. “Self-organized coordinated motion in groups of physically connected robots.” *IEEE Transactions on Systems Man and Cybernetics-Part B-Cybernetics*, **37**(1):224–239, 2007.



- [BV02] M. Bowling and M. Veloso. “Multiagent learning using a variable learning rate.” *Artificial Intelligence*, **136**(2):215–250, 2002.
- [CB98] C. Claus and C. Boutilier. “The dynamics of reinforcement learning in cooperative multiagent systems.” In *Proceedings of the National Conference on Artificial Intelligence*, pp. 746–752. John Wiley & Sons LTD, 1998.
- [CBM10] Jeff Clune, Benjamin E. Beckmann, Philip K. McKinley, and Charles Ofria. “Investigating whether HyperNEAT produces modular neural networks.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*, New York, NY, 2010. ACM Press.
- [CBO09] Jeff Clune, Benjamin E. Beckmann, Charles Ofria, and Robert T. Pennock. “Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding.” In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Session on Evolutionary Robotics*, Piscataway, NJ, USA, 2009. IEEE Press.
- [CBP09] Jeff Clune, Benjamin E. Beckmann, Robert T. Pennock, and Charles Ofria. “HybrID: A Hybridization of Indirect and Direct Encodings for Evolutionary Computation.” In *Proceedings of the European Conference on Artificial Life (ECAL-2009)*,, 2009.
- [Chu86] Paul M. Churchland. “Some reductive strategies in cognitive neurobiology.” *Mind*, **95**:279–309, 1986.

- [CK04] Dmitri B. Chklovskii and Alexei A. Koulakov. “MAPS IN THE BRAIN: What Can We Learn From Them?” *Annual Review of Neuroscience*, **27**:369–392, 2004.
- [COP08] Jeff Clune, Charles Ofria, and Robert T. Pennock. “How a Generative Encoding Fares as Problem-regularity Decreases.” In *Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN 2008)*, pp. 258–367, Berlin, 2008. Springer.
- [CPO09] Jeff Clune, Robert T. Pennock, and Charles Ofria. “The Sensitivity of HyperNEAT to Different Geometric Representations of a Problem.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)*, New York, NY, USA, 2009. ACM Press.
- [CS07] V. Conitzer and T. Sandholm. “AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents.” *Machine Learning*, **67**(1):23–43, 2007.
- [CW10] Adam Campbell and Annie S. Wu. “Multi-agent role allocation: issues, approaches, and multiple perspectives.” *Autonomous Agents and Multi-Agent Systems*, pp. 1–39, 2010.
- [DAR05] DARPA. “Transfer learning proposer information pamphlet (baa # 05-29).” <http://www.darpa.mil/ipto/solicitations/closed/05-29PIP.htm>, 2005.
- [Daw86] R. Dawkins. *The Blind Watchmaker*. Longman, Essex, U.K., 1986.

- [DB96] F. Dellaert and R. D. Beer. “A Developmental Model for the Evolution of Complete Autonomous Agents.” In Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W. Wilson, editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. Cambridge, MA: MIT Press, 1996.  
*[Conference held September 9-13, 1996.]*
- [Del95] Frank Dellaert. “*Toward a Biologically Defensible Model of Development.*”. Master’s thesis, Case Western Reserve University, Cleveland, OH, 1995.
- [DKS09] J. Drchal, J. Koutnk, and M. Snorek. “HyperNEAT Controlled Robots Learn to Drive on Roads in Simulated Environment.” In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009)*, Piscataway, NJ, USA, 2009. IEEE Press.
- [DLR10] David D’Ambrosio, Joel Lehman, Sebastian Risi, and Kenneth O. Stanley. “Evolving Policy Geometry for Scalable Multiagent Learning.” In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)*, pp. 731–738. International Foundation for Autonomous Agents and Multiagent System, 2010.
- [DS07] David D’Ambrosio and Kenneth O. Stanley. “A Novel Generative Encoding for Exploiting Neural Network Sensor and Output Geometry.” In *Proceedings of the*

- Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.
- [DS08] David B. D’Ambrosio and Kenneth O. Stanley. “Generative Encoding for Multiagent Learning.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, New York, NY, 2008. ACM Press.
- [DSG98] P. Deloukas, G. D. Schuler, G. Gyapay, E. M. Beasley, C. Soderlund, P. Rodriguez-Tome, L. Hui, T. C. Matise, K. B. McKusick, J. S. Beckmann, S. Bentolila, M. Bihoreau, B. B. Birren, J. Browne, A. Butler, A. B. Castle, N. Chianilkulchai, C. Clee, P. J. Day, A. Dehejia, T. Dibling, N. Drouot, S. Duprat, C. Fizames, and D. R. Bentley. “A Physical Map of 30,000 Human Genes.” *Science*, **282**(5389):744–746, October 23 1998.
- [Dup90] Trevor Nevitt Dupuy. *The Evolution of Weapons and Warfare*. Da Capo, New York, NY, USA, 1990.
- [Egg97a] P. Eggenberger. “Evolving Morphologies of Simulated 3d Organisms Based on Differential Gene Expression.” *Fourth European Conference on Artificial Life*, 1997.
- [Egg97b] P. Eggenberger. “Evolving Morphologies of Simulated 3D Organisms Based on Differential Gene Expression.” In Phil Husbands and Inman Harvey, editors, *Proceedings of the Fourth European Conference on Artificial Life*, pp. 205–213. Cambridge, MA: MIT Press, 1997.

- [FDM08] Dario Floreano, Peter Dürri, and Claudio Mattiussi. “Neuroevolution: from architectures to learning.” *Evolutionary Intelligence*, **1**:47–62, 2008.
- [Fed04a] Diego Federici. “Evolving a Neurocontroller Through a Process of Embryogeny.” In Stefan Schaal, Auke Jan Ijspeert, Aude Billard, Sethu Vijayakumar, John Halam, and Jean-Arcady, editors, *Proceedings of the Eighth International Conference on Simulation and Adaptive Behavior (SAB-2004)*, pp. 373–384, Cambridge, MA, 2004. MIT Press.
- [Fed04b] Diego Federici. “Using Embryonic Stages to Increase the Evolvability of Development.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004) Workshop Program*, Berlin, 2004. Springer Verlag.
- [FP00] S.G. Ficici and J.B. Pollack. “A game-theoretic approach to the simple coevolutionary algorithm.” *Lecture notes in computer science*, pp. 467–476, 2000.
- [FU00] D. Floreano and J. Urzelai. “Evolutionary Robots with On-Line Self-Organization and Behavioral Fitness.” *Neural Networks*, **13**:431–4434, 2000.
- [GC02] Geoffrey J. Goodhill and Miguel A. Carreira-Perpinn. “Cortical Columns.” In L. Nadel, editor, *Encyclopedia of Cognitive Science*, volume 1, pp. 845–851. MacMillan Publishers Ltd., London, 2002.
- [GM97] Faustino Gomez and Risto Miikkulainen. “Incremental Evolution of Complex General Behavior.” *Adaptive Behavior*, **5**:317–342, 1997.

- [GM99] Faustino Gomez and Risto Miikkulainen. “Solving Non-Markovian Control Tasks with Neuroevolution.” In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pp. 1356–1361, San Francisco, 1999. Kaufmann.
- [Gre06] Colin Green. “SharpNEAT homepage.” <http://sharpneat.sourceforge.net/>, 2003–2006.
- [Gru94] Frederic Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm*. PhD thesis, Ecole Normale Supérieure de Lyon, France, 1994.
- [GS07] Jason Gauci and Kenneth O. Stanley. “Generating Large-Scale Neural Networks Through Discovering Geometric Regularities.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.
- [GS08] Jason Gauci and Kenneth O. Stanley. “A Case Study on the Critical Role of Geometric Regularity in Machine Learning.” In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, Menlo Park, CA, 2008. AAAI Press.
- [GS10a] Jason Gauci and Kenneth O. Stanley. “Autonomous Evolution of Topographic Regularities in Artificial Neural Networks.” *Neural Computation*, **22**(7):1860–1898, 2010.

- [GS10b] Jason Gauci and Kenneth O. Stanley. “Indirect Encoding of Neural Networks for Scalable Go.” In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature – PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pp. 354–363. Springer, 2010.
- [GW92] C. D. Gilbert and T. N. Wiesel. “Receptive Field Dynamics in Adult Primary Visual Cortex.” *Nature*, **356**:150–152, March 1992.
- [GWP96] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. “A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks.” In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 81–89, Cambridge, MA, 1996. MIT Press.
- [Har93] Inman Harvey. *The Artificial Evolution of Adaptive Behavior*. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex, Sussex, 1993.
- [HGS09a] Erin Hastings, Ratan Guha, and Kenneth O. Stanley. “Evolving Content in the Galactic Arms Race Video Game.” In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG-09)*, Piscataway, NJ, 2009. IEEE Press.
- [HGS09b] Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley. “Interactive Evolution of Particle Systems for Computer Graphics and Animation.” In *IEEE Transactions on Evolutionary Computation*, Piscataway, NJ, 2009. IEEE Press.

- [HP02] Gregory S. Hornby and Jordan B. Pollack. “Creating High-Level Components with a Generative Representation for Body-Brain Evolution.” *Artificial Life*, **8**(3), 2002.
- [HRS08] Amy K. Hoover, Michael P. Rosario, and Kenneth O. Stanley. “Scaffolding for Interactively Evolving Novel Drum Tracks for Existing Songs.” In Mario Giacobini, editor, *Proceedings of the Sixth European Workshop on Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMUSART 2008)*, pp. 412–422. Springer, March 2008.
- [HS96] T.D. Haynes and S. Sen. “Co-adaptation in a team.” *International Journal of Computational Intelligence and Organizations*, **1**(4):1–20, 1996.
- [HS09] Amy K. Hoover and Kenneth O. Stanley. “Exploiting Functional Relationships in Musical Composition.” *Connection Science Special Issue on Music, Brain, and Cognition*, **21**(2 and 3):227–251, 2009.
- [HW65] David H. Hubel and Torsten N. Wiesel. “Receptive Fields and Functional Architecture in Two Nonstriate Visual Areas (18 and 19) of the Cat.” *Journal of Neurophysiology*, **28**:229–289, 1965.
- [HW98] Junling Hu and Michael P. Wellman. “Multiagent reinforcement learning: theoretical framework and an algorithm.” In *Proc. 15th International Conf. on Machine Learning*, pp. 242–250. Morgan Kaufmann, San Francisco, CA, 1998.



- [HW03] Junling Hu and Michael P. Wellman. “Nash Q-learning for general-sum stochastic games.” *The Journal of Machine Learning Research*, **4**:1039–1069, 2003.
- [ICM94] Michael L. Littman. “Markov Games as a Framework for Multi-Agent Reinforcement Learning.” In *Machine Learning: Proceedings of the 11th Annual Conference*, pp. 157–163. San Francisco: Kaufmann, 1994.
- [ISK03] Y. Ishiwaka, T. Sato, and Y. Kakazu. “An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning.” *Robotics and Autonomous Systems*, **43**(4):245–256, 2003.
- [Jak95] Nick Jakobi. “Harnessing Morphogenesis.” In *Proceedings of Information Processing in Cells and Tissues*, pp. 29–41, University of Liverpool, 1995.
- [JG00] K.C. Jim and C.L. Giles. “Talking helps: Evolving communicating agents for the predator-prey pursuit problem.” *Artificial Life*, **6**(3):237–254, 2000.
- [KGM10] D.B. Knoester, H.J. Goldsby, and P.K. McKinley. “Neuroevolution of mobile ad hoc networks.” In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 603–610. ACM, 2010.
- [KHB05] J.R. Kok, P. Hoen, B. Bakker, and N. Vlassis. “Utile coordination: Learning interdependencies among cooperative agents.” In *Proc. Symp. on Computational Intelligence and Games*, pp. 29–36. Citeseer, 2005.

- [KR01] Maciej Komosinski and Adam Rotaru-Varga. “Comparison of Different Genotype Encodings for Simulated 3D Agents.” *Artificial Life*, **7**(4):395–418, 2001.
- [KSJ91] Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell, editors. *Principles of Neural Science*. Elsevier, Amsterdam, third edition, 1991.
- [KSM06] Nate Kohl, Kenneth Stanley, Risto Miikkulainen, Michael Samples, and Rini Sherony. “Evolving a Real-World Vehicle Warning System.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)*, pp. 1681–1688, July 2006.
- [KUP03] E. Kutschinski, T. Uthmann, and D. Polani. “Learning competitive pricing strategies by multi-agent reinforcement learning.” *Journal of Economic Dynamics and Control*, **27**(11-12):2207–2218, 2003.
- [Lin68] A. Lindenmayer. “Mathematical Models for Cellular Interaction in Development Parts I and II.” *Journal of Theoretical Biology*, **18**:280–299 and 300–315, 1968.
- [Lin74] A. Lindenmayer. “Adding Continuous Components to L-systems.” In G. Rozenberg and A. Salomaa, editors, *L Systems, Lecture Notes in Computer Science 15*, pp. 53–68. Springer-Verlag, Heidelberg, Germany, 1974.
- [LS96] Sean Luke and Lee Spector. “Evolving Graphs and Networks with Edge Encoding: Preliminary Report.” In J. R. Koza, editor, *Late-Breaking Papers of Genetic Programming 1996*. Stanford Bookstore, 1996.

- [Mar99] Andrew P. Martin. “Increasing Genomic Complexity by Gene Duplication and the Origin of Vertebrates.” *The American Naturalist*, **154**(2):111–128, 1999.
- [Mat97] M.J. Matarić. “Reinforcement learning in the multi-robot domain.” *Autonomous Robots*, **4**(1):73–83, 1997.
- [MBC10] J.-S. Marier, C. Besse, and B. Chaib-draa. “Solving the continuous time multi-agent patrol problem.” In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 941 –946, May 2010.
- [Mic03] Thomas Miconi. “When Evolving Populations is Better than Coevolving Individuals: The Blind Mice Problem.” In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI ’03)*. Morgan Kaufmann, 2003.
- [Mil04] Julian F. Miller. “Evolving a Self-Repairing, Self-Regulating, French Flag Organism.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Berlin, 2004. Springer Verlag.
- [MMG01] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. “Hierarchical multi-agent reinforcement learning.” In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS ’01, pp. 246–253, New York, NY, USA, 2001. ACM.

- [MSR91] Eric Mjolsness, David H. Sharp, and John Reinitz. “A Connectionist Model of Development.” *Journal of Theoretical Biology*, **152**:429–453, 1991.
- [Nol97] Stefano Nolfi. “Using Emergent Modularity to Develop Control Systems for Mobile Robotics.” *Adaptive Behavior*, **3–4**:343–364, 1997.
- [PB99] B. Price and C. Boutilier. “Implicit imitation in multiagent reinforcement learning.” In *Machine Learning-International Workshop then Conference-*, pp. 325–334. Morgan Kaufmann Publishers, INC., 1999.
- [PD94] M.A. Potter and K.A. De Jong. “A cooperative coevolutionary approach to function optimization.” *Lecture Notes in Computer Science*, **866**:249–259, 1994.
- [PDG95] Mitchell A. Potter, Kenneth A. De Jong, and John J. Grefenstette. “A Coevolutionary Approach to Learning Sequential Decision Rules.” In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*. San Francisco: Kaufmann, 1995.
- [PL90] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, Heidelberg, Germany, 1990.
- [PL05] Liviu Panait and Sean Luke. “Cooperative Multi-Agent Learning: The State of the Art.” *Autonomous Agents and Multi-Agent Systems*, **3**(11):383–434, November 2005.

- [PLH06] Liviu Panait, Sean Luke, and Joseph F. Harrison. “Archive-based cooperative co-evolutionary algorithms.” In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 345–352. ACM New York, NY, USA, 2006.
- [PLW06] Liviu Panait, Sean Luke, and Rudolf Paul Wiegand. “Biasing coevolutionary search for optimal multiagent behaviors.” *IEEE Transactions on Evolutionary Computation*, **10**(6):629–645, 2006.
- [PMS01] M.A. Potter, L.A. Meeden, and A.C. Schultz. “Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists.” In *International Joint Conference on Artificial Intelligence*, pp. 1337–1343. Lawrence Erlbaum Associates LTD, 2001.
- [PSG98] N. Puppala, S. Sen, and M. Gordin. “Shared memory based cooperative coevolution.” In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pp. 570–574, 1998.
- [PT09] Scott Proper and Prasad Tadepalli. “Solving multiagent assignment markov decision processes.” In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 681–688. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

- [PTL08] Liviu Panait, Karl Tuyls, and Sean Luke. “Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective.” *The Journal of Machine Learning Research*, **9**:423–457, 2008.
- [PWL03] L. Panait, R.P. Wiegand, and S. Luke. “Improving coevolutionary search for optimal multiagent behaviors.” *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 653–658, 2003.
- [Raf96] Rudolf A. Raff. *The Shape of Life: Genes, Development, and the Evolution of Animal Form*. The University of Chicago Press, Chicago, 1996.
- [RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning Internal Representations by Error Propagation.” In *Parallel Distributed Processing*, pp. 318–362. MIT Press, 1986.
- [RS10] Sebastian Risi and Kenneth O. Stanley. “Indirectly Encoding Neural Plasticity as a Pattern of Local Rules.” In *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior (SAB2010)*, Berlin, 2010. Springer.
- [RVH09] Sebastian Risi, Sandy D. Vanderbleek, Charles E. Hughes, and Kenneth O. Stanley. “How Novelty Search Escapes the Deceptive Trap of Learning to Learn.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)*, New York, NY, USA, 2009. ACM Press.

- [RW03] Z. Ren and A.B. Williams. “Lessons learned in single-agent and multiagent learning with robot foraging.” In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 3, pp. 2757–2762 vol.3, Oct. 2003.
- [SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. The MIT press, 1998.
- [SBD08] Jimmy Secretan, Nicholas Beato, David B. D’Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O. Stanley. “Picbreeder: Evolving Pictures Collaboratively Online.” In *CHI ’08: Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pp. 1759–1768, New York, NY, USA, 2008. ACM.
- [SBM05a] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. “Evolving Neural Network Agents in the NERO Video Game.” In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games*, 2005.
- [SBM05b] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. “Real-Time Neuroevolution in the NERO Video Game.” *IEEE Transactions on Evolutionary Computation Special Issue on Evolutionary Computation and Games*, **9**(6):653–668, 2005.
- [SDG09] Kenneth O. Stanley, David B. D’Ambrosio, and Jason Gauci. “A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks.” *Artificial Life*, **15**(2):185–212, 2009.

- [SF95] N. Saravanan and David B. Fogel. “Evolving Neural Control Systems.” *IEEE Expert*, pp. 23–27, June 1995.
- [SH02] N. Suematsu and A. Hayashi. “A multiagent reinforcement learning algorithm using extended optimal response.” In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pp. 370–377. ACM New York, NY, USA, 2002.
- [Sim94a] Karl Sims. “Evolving 3D Morphology and Behavior by Competition.” In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV)*, pp. 28–39. MIT Press, Cambridge, MA, 1994.
- [Sim94b] Karl Sims. “Evolving 3D Morphology and Behavior by Competition.” In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems (Artificial Life IV)*, pp. 28–39. Cambridge, MA: MIT Press, 1994.  
*[Conference held July 6-8, 1994. Second printing was in 1995.]*
- [SK08] A. Servin and D. Kudenko. “Multi-Agent Reinforcement Learning for Intrusion Detection.” *Lecture Notes in Computer Science*, **4865**:211, 2008.
- [SKM00] S. Singh, M. Kearns, and Y. Mansour. “Nash convergence of gradient dynamics in general-sum games.” In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000.



- [SKM05] Kenneth O. Stanley, Nate Kohl, and Risto Miikkulainen. “Neuroevolution of an Automobile Crash Warning System.” In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2005.
- [SM02a] Kenneth O. Stanley and Risto Miikkulainen. “Efficient Reinforcement Learning Through Evolving Neural Network Topologies.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, 2002. Kaufmann.
- [SM02b] Kenneth O. Stanley and Risto Miikkulainen. “Evolving Neural Networks Through Augmenting Topologies.” *Evolutionary Computation*, **10**:99–127, 2002.
- [SM03] Kenneth O. Stanley and Risto Miikkulainen. “A Taxonomy for Artificial Embryogeny.” *Artificial Life*, **9**(2):93–130, 2003.
- [SM04a] Kenneth O. Stanley and Risto Miikkulainen. “Competitive Coevolution Through Evolutionary Complexification.” *Journal of Artificial Intelligence Research*, **21**:63–100, 2004.
- [SM04b] Kenneth O. Stanley and Risto Miikkulainen. “Evolving a Roving Eye for Go.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Berlin, 2004. Springer Verlag.

- [SPG04] Y. Shoham, R. Powers, and T. Grenager. “Multi-agent reinforcement learning: a critical survey.” In *AAAI Fall Symposium on Artificial Multi-Agent Learning*, 2004.
- [SRC04] Hugo Santana, Geber Ramalho, Vincent Corruble, and Bohdana Ratitch. “Multi-Agent Patrolling with Reinforcement Learning.” *Autonomous Agents and Multi-agent Systems, International Joint Conference on*, **3**:1122–1129, 2004.
- [SS01] Peter Stone and Richard S. Sutton. “Scaling Reinforcement Learning toward RoboCup Soccer.” In *Proc. 18th International Conf. on Machine Learning*, pp. 537–544. Morgan Kaufmann, San Francisco, CA, 2001.
- [SSK05] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. “Reinforcement learning for robocup soccer keepaway.” *Adaptive Behavior*, **13**(3):165, 2005.
- [SSS01] Peter Stone, Richard S. Sutton, and Satinder P. Singh. “Reinforcement Learning for 3 vs. 2 Keepaway.” In *RoboCup 2000: Robot Soccer World Cup IV*, pp. 249–258, London, UK, 2001. Springer-Verlag.
- [Sta06] Kenneth O. Stanley. “Exploiting Regularity Without Development.” In *Proceedings of the AAAI Fall Symposium on Developmental Systems*, Menlo Park, CA, 2006. AAAI Press.

- [Sta07] Kenneth O. Stanley. “Compositional Pattern Producing Networks: A Novel Abstraction of Development.” *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, **8**(2):131–162, 2007.
- [Sut96] Richard Sutton. “Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding.” In *Advances in Neural Information Processing Systems 8*, pp. 1038–1044, 1996.
- [Sut09] Richard S. Sutton. “Tile Coding Software, Version 2.0.”, December 2009. <http://webdocs.cs.ualberta.ca/~sutton/tiles2.html>.
- [SV00] Peter Stone and Manuela Veloso. “Multiagent Systems: A Survey from a Machine Learning Perspective.” *Autonomous Robots*, **8**(3):345–383, 2000.
- [Tan97] M. Tan. “Multi-agent reinforcement learning: Independent vs. cooperative agents.” *Readings in agents*, pp. 487–494, 1997.
- [TOL08] L. Trujillo, G. Olague, E. Lutton, and F.F. de Vega. “Discovering Several Robot Behaviors through Speciation.” *Applications of Evolutionary Computing: Evoworkshops 2008: Evocomnet, Evofin, Evohot, Evoiasp, Evomusart, Evonum, Evostoc, and Evotranslog*, p. 164, 2008.
- [Tur52] Alan Turing. “The Chemical Basis of Morphogenesis.” *Philosophical Transactions of the Royal Society B*, **237**:37–72, 1952.

- [TWS06] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. “Comparing Evolutionary and Temporal Difference Methods in a Reinforcement Learning Domain.” In *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1321–1328, July 2006.
- [VS10a] Phillip Verbancsics and Kenneth O. Stanley. “Evolving Static Representations for Task Transfer.” *Journal of Machine Learning Research (JMLR)*, **11**:1737–1769, 2010.
- [VS10b] Phillip Verbancsics and Kenneth O. Stanley. “Task Transfer through Indirect Encoding.” In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, New York, NY, 2010. ACM Press.
- [WB10] Samuel J. Waskow and Ana L. C. Bazzan. “Improving space representation in multiagent learning via tile coding.” In *Proceedings of the 20th Brazilian conference on Advances in artificial intelligence, SBIA’10*, pp. 153–162, Berlin, Heidelberg, 2010. Springer-Verlag.
- [Web] Webots. “<http://www.cyberbotics.com>.” Commercial Mobile Robot Simulation Software.
- [WHR87] James D. Watson, Nancy H. Hopkins, Jeffrey W. Roberts, Joan A. Steitz, and Alan M. Weiner. *Molecular Biology of the Gene Fourth Edition*. The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

- [Wie04] Rudolf Paul Wiegand. *An analysis of cooperative coevolutionary algorithms*. PhD thesis, George Mason University, Fairfax, VA, USA, 2004. Director-Kenneth A. Jong.
- [WKF09] Markus Waibel, Laurent Keller, and Dario Floreano. “Genetic Team Composition and Level of Selection in the Evolution of Multi- Agent Systems.” *IEEE Transactions on Evolutionary Computation*, 2009.
- [WM97] David H. Wolpert and William Macready. “No Free Lunch Theorems for Optimization.” *IEEE Transactions on Evolutionary Computation*, 1:67–82, 1997.
- [Yao99] Xin Yao. “Evolving Artificial Neural Networks.” *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
- [YM07] C. Yong and R. Miikkulainen. “Coevolution of Role-Based Cooperation in Multi-Agent Systems.” Technical Report AI07-338, The University of Texas at Austin Department of Computer Sciences, 2007.
- [ZBL99] Michael J. Zigmond, Floyd E. Bloom, Story C. Landis, James L. Roberts, and Larry R. Squire, editors. *Fundamental Neuroscience*. Academic Press, London, 1999.